# R Training Documentation

2023-10-17

## Introduction

### Source files, directories, important functions in R

The working directory is the folder your R session 'lives in.' In general, it is good practice to have your working directory set to the folder which houses the open file. getwd() displays the current working directory of the session. You can set the working directory to the path of the desired folder by using setwd(...).

If ever you are confused about a package or function in RStudio, typing '?' in front of the argument will display the documentation for that function in your session. For information on VectorSurv specific functions, please see the documentation below.

In the code, a '#' tells the compiler to ignore that line. This can be used to add comments or to intentionally ignore code.

```
#To get current directory
getwd()
```

```
## [1] "C:/Users/Christina/Desktop/R-workshop"
```

```
#To get information on function in R documentation
?getwd()


#'VS_functions.R' should be in the same directory as the .rmd
#'files which access its functions, you can set the working directory

#Path to folder on my computer: "C:\Users\Christina\Desktop\R-tutorial"
#this path will be slightly different for you depending on the file location.

#Set below to the location of the downloaded files on your machine,
#make sure to change the direction of the slash from "\" to "/" to indicate opening of the folder.

setwd("/Users/Christina/Desktop/R-workshop")
```

There is one .R file. The .R file is a source file that contains all of the functions you will need to run the .RMarkdown files. You do not need to touch the source file. The .RMarkdown files are pre-coded sample reports that you can customize to suit your needs.

```
#loads relevant packages and functions
source("source_functions.R")
```

# Retrieving Data

### getToken()

*Description*

getToken() returns a token needed to run getArthroCollections() and getPools(). The function prompts users for their Gateway credentials. If credentials are accepted, the function returns a user token needed to obtain data.

*Usage*

getToken()

*Arguments*

```r
token = getToken()
```

### getArthroCollections(. . . )

*Description*

getArthroCollections(. . . ) obtains collections data on a year range (start_year, end_year). It prompts the user for their Gateway username and password before retrieving the associated data. You can only retrieve data from agencies linked to your Gateway account.

*Usage*

getArthroCollections(token,start_year, end_year)

*Arguments*

- token: access token retrived from getToken()
- start_year: Beginning of year range
- end_year: End of year range

```r
#Example
collections = getArthroCollections(token, 2022,2023)
```

### getPools(. . . )

*Description*

getPools(. . . ) similar to getArthroCollections() obtains pools on a year range (start_year, end_year). It prompts the user for their Gateway username and password before retrieving the associated data. getPools() retrieve data for both mosquito and tick pools.

*Usage*

getPools(token, start_year, end_year)

*Arguments*

- token: access token retrived from getToken()
- start_year: Beginning of year range
- end_year: End of year range

```r
#Example
pools = getPools(token, 2022,2023)
```

## Write Data to file

You can save retrieved data as a .csv file in your current directory using write.csv(). That same data can be retrieved using read.csv(). Writing data to a .csv can make the rendering process more efficient when generating reports in R. We recommend that you write the data pulled from our API into a csv and then load that data when generating reports.

read.csv(…)

```
#creates a file named "collections_18_23.csv" in your current directory
 write.csv(x = collections, file = "collections_22_23.csv")

#loads collections data
collections = read.csv("collections_22_23.csv")
```

## Basic subseting, filtering, grouping, pivoting

Data can be subset to contain columns of interest. Sub-setting can also be used to reorder the columns in a dataframe.Do not subset collections or pools data before inputting them into VectorSurv calculator functions to avoid losing essential columns. It is recommended to subset after calculations are complete and before inputting into a table generator. **Remember, subseting, filtering, grouping and summarizing will not change the value of the data unless it is reassigned to the same variable name.** We recomment creating a new variable for processed data.

### Subseting

```
#Subset using column names or index number

colnames(collections) #displays column names and associated index
```

```
##  [1] "X"                         "collection_id"
##  [3] "collection_num"            "collection_date"
##  [5] "collection_date_date_only" "comments"
##  [7] "identified_by"             "num_trap"
##  [9] "site"                      "surv_year"
## [11] "trap_nights"               "trap_problem_bit"
## [13] "user"                      "add_date"
## [15] "deactive_date"             "updated"
## [17] "id"                        "num_count"
## [19] "sex_id"                    "sex_type"
## [21] "sex_name"                  "species_id"
## [23] "species_full_name"         "species_display_name"
## [25] "agency_id"                 "agency_code"
## [27] "agency_name"               "trap_id"
## [29] "trap_acronym"              "trap_name"
## [31] "trap_presence"
```

```
#Subseting by name
head(collections[c("collection_date","species_display_name","num_count")])
```

```
##            collection_date species_display_name num_count
## 1 2023-12-20T08:00:00.000Z           Cx pipiens         2
## 2 2023-12-20T08:00:00.000Z           Cx pipiens         2
## 3 2023-12-20T08:00:00.000Z           Cx pipiens         1
## 4 2023-12-20T08:00:00.000Z           Cx pipiens         1
## 5 2023-12-20T08:00:00.000Z           Cx pipiens         1
## 6 2023-12-20T08:00:00.000Z           Cx pipiens         3
```

```
#by index
head(collections[c(3,23,17)])
```

```
##   collection_num species_full_name      id
## 1           9224     Culex pipiens 6283795
## 2           9224     Culex pipiens 6283794
## 3           9225     Culex pipiens 6283816
## 4           9226     Culex pipiens 6283835
## 5           9226     Culex pipiens 6283834
## 6           9227     Culex pipiens 6283897
```

```
#to save a subset
collections_subset = collections[c(3,23,17)]
```

#### Filtering and subsetting in dplyr

Dplyr is a powerful package for filtering and sub-setting data. It follows logic similar to SQL queries.

For more information on data manipulation using dplyr Click Here

Dplyr utilizes the pipe operator '%>%' to send data into functions. The head() function returns the first few rows of data, specifying head(1) tells the software to return only the first row for viewing purposes. Remove head() to see all the data or reassign the data to a new variable.

```
#Subsetting columns with dplyr 'select'
collections %>%
  select(collection_date,species_display_name,num_count)%>%
  head()
```

```
##            collection_date species_display_name num_count
## 1 2023-12-20T08:00:00.000Z           Cx pipiens         2
## 2 2023-12-20T08:00:00.000Z           Cx pipiens         2
## 3 2023-12-20T08:00:00.000Z           Cx pipiens         1
## 4 2023-12-20T08:00:00.000Z           Cx pipiens         1
## 5 2023-12-20T08:00:00.000Z           Cx pipiens         1
## 6 2023-12-20T08:00:00.000Z           Cx pipiens         3
```

Below are more examples for filtering data.

```
#filtering with dplyr 'filter'
collections_pip = collections %>%
  filter(species_display_name=="Cx pipiens")

#filtering multiple arguments using '%in%'
```

```r
collections_pip_tar = collections %>%
  filter(species_display_name %in% c("Cx pipiens", "Cx tarsalis"),
         site %in% c(119819,102832)) #filters site codes
```

### Group by

In addition to filtering and sub-setting, data can be group by variables and summarized.

```r
#groups by species and collection date and sums the number counted

collections %>%
  group_by(collection_date, species_display_name) %>%
  summarise(sum_count = sum(num_count, na.rm=T))%>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   collection_date [1]
##   collection_date          species_display_name sum_count
##   <chr>                    <chr>                    <int>
## 1 2022-01-04T08:00:00.000Z An freeborni                 1
## 2 2022-01-04T08:00:00.000Z Cs incidens                  2
## 3 2022-01-04T08:00:00.000Z Cs inornata                  2
## 4 2022-01-04T08:00:00.000Z Cx pipiens                  22
## 5 2022-01-04T08:00:00.000Z Cx stigmatosoma              2
## 6 2022-01-04T08:00:00.000Z Cx tarsalis                  1
```

```r
#groups by species and collection date and takes the average the number counted

collections %>%
  group_by(collection_date, species_display_name) %>%
  summarise(avg_count = mean(num_count, na.rm=T))%>%
  head()
```

```
## # A tibble: 6 x 3
## # Groups:   collection_date [1]
##   collection_date          species_display_name avg_count
##   <chr>                    <chr>                    <dbl>
## 1 2022-01-04T08:00:00.000Z An freeborni              1
## 2 2022-01-04T08:00:00.000Z Cs incidens               1
## 3 2022-01-04T08:00:00.000Z Cs inornata               1
## 4 2022-01-04T08:00:00.000Z Cx pipiens                2.44
## 5 2022-01-04T08:00:00.000Z Cx stigmatosoma           1
## 6 2022-01-04T08:00:00.000Z Cx tarsalis               1
```

### Pivoting

Data can be manipulated into long and wide (spread sheet) forms using pivot_wider and pivot_longer. By default data from the API is in long form. Here we pivot on species and sex condition names using num_count as values. The end result is data with num_count values in the columns named species_sex. For more on pivoting see ??pivot_longer and ??pivot_wider.

```r
collections_wide = pivot_wider(collections,
                               names_from = c("species_display_name","sex_name"),
                               values_from = "num_count")
```

# Calculations

## Abundance

### getAbundance(…)

*Description*

getAbundance(…) uses any amount of arthro collections data to calculate the abundance for the specified parameters. The function calculates using the methods of the Gateway Abundance calculator.

*Usage*

getAbundance(collections,interval, species_list = NULL, trap_list = NULL, species_seperate = FALSE)

*Arguments*

- collections: Collections data retrieved from getArthroCollections(…)
- interval: Calculation interval for abundance, accepts "collection_date","Biweek","Week", and "Month.
- species_list: Species filter for calculating abundance. Species_display_name is the accepted notation. To see a list of species present in your data run unique(collections$species_display_name). If species is unspecified, the default NULL will return data for all species in data.
- trap_list: Trap filter for calculating abundance. Trap_acronym is the is the accepted notation. Run unique(collections$trap_acronym) to see trap types present in your data. If trap_list is unspecified, the default NULL will return data for all trap types.
- species_seperate: Should the species in species_list have abundance calculated separately? Setting to FALSE calculates the combined abundance. The same result can be performed by calculating on one species at the time.

```r
getAbundance(collections,
             interval = "Biweek",
             species_list = c("Cx tarsalis", "Cx pipiens"),
             trap_list = "CO2",
             species_seperate = FALSE)
```

```
##    EPIYEAR Biweek Count Trap_Events Abundance
## 1     2023     10   882          65     13.57
## 2     2023     11  3254         142     22.92
## 3     2023     12  4395         153     28.73
## 4     2023     13 15803         182     86.83
## 5     2023     14 24939         226    110.35
## 6     2023     15 24113         217    111.12
## 7     2023     16 19062         255     74.75
## 8     2023     17 12865         226     56.92
## 9     2023     18 10088         213     47.36
## 10    2023     19  7161         211     33.94
## 11    2023     20  5934         211     28.12
## 12    2023     21  2806         144     19.49
## 13    2023     22   279          74      3.77
```

```
## 14    2023    23   122          43      2.84
## 15    2023    24    29          33      0.88
## 16    2022     4     9           3      3.00
## 17    2022     9  1358         126     10.78
## 18    2022    10  1202         133      9.04
## 19    2022    11  1969         145     13.58
## 20    2022    12  3503         159     22.03
## 21    2022    13  5630         159     35.41
## 22    2022    14 10444         154     67.82
## 23    2022    15  9722         178     54.62
## 24    2022    16  7949         186     42.74
## 25    2022    17  6501         180     36.12
## 26    2022    18  6038         166     36.37
## 27    2022    19  3798         163     23.30
## 28    2022    20  1869         120     15.57
## 29    2022    21  1189          84     14.15
```

## Abundance Anomaly (comparison to 5 year average)

### getAbundanceAnomaly()

*Description*

getAbundanceAnomaly(..) requires at least five years prior to the target_year of arthro collections data to calculate for the specified parameters. The function uses the methods of the Gateway Abundance Anomaly calculator, and will not work if there is fewer than five years of data present.

*Usage*

getAbundanceAnomaly(collections,interval,target_year, species_list = NULL, trap_list = NULL, species_seperate = FALSE)

*Arguments*

- collections: Collections data retrieved from getArthroCollections(…)
- interval: Calculation interval for abundance, accepts "collection_date","Biweek","Week", and "Month.
- target_year: Year to calculate analysis on. Collections data must have a year range of at least (target_year - 5, target_year).
- species_list: Species filter for calculating abundance. Species_display_name is the accepted notation. To see a list of species present in your data run unique(collections$species_display_name). If species is unspecified, the default NULL will return data for all species in data.
- trap_list: Trap filter for calculating abundance. Trap_acronym is the is the accepted notation. Run unique(collections$trap_acronym) to see trap types present in your data. If trap_list is unspecified, the default NULL will return data for all trap types.
- species_seperate: Should the species in species_list have abundance calculated separately? Setting to FALSE calculates the combined abundance. The same result can be performed by calculating on one species at the time.

```r
collections_18_23 = getArthroCollections(token, 2018,2023)

getAbundanceAnomaly(collections_18_23,
                    interval = "Biweek",
                    target_year = 2023,
                    species_list = c("Cx tarsalis", "Cx pipiens"),
                    trap_list = "CO2",
                    species_seperate = FALSE)
```

```
##    Biweek EPIYEAR Count Trap_Events Abundance Five_Year_Avg  Delta
## 1      10    2023   882          65     13.57        12.926   4.98
## 2      11    2023  3254         142     22.92        19.666  16.55
## 3      12    2023  4395         153     28.73        37.988 -24.37
## 4      13    2023 15803         182     86.83        54.496  59.33
## 5      14    2023 24939         226    110.35        81.972  34.62
## 6      15    2023 24113         217    111.12        75.588  47.01
## 7      16    2023 19062         255     74.75        78.528  -4.81
## 8      17    2023 12865         226     56.92        66.406 -14.28
## 9      18    2023 10088         213     47.36        61.704 -23.25
## 10     19    2023  7161         211     33.94        51.736 -34.40
## 11     20    2023  5934         211     28.12        32.970 -14.71
## 12     21    2023  2806         144     19.49        20.082  -2.95
## 13     22    2023   279          74      3.77        18.330 -79.43
## 14     23    2023   122          43      2.84         1.730  64.16
```

## Infection Rate

### getInfectionRate()

*Description*

getInfectionRate(..) requires at least five years prior to the target_year of arthro collections data to calculate for the specified parameters. The function uses the methods of the Gateway Abundance Anomaly calculator, and will not work if there is fewer than five years of data present.

*Usage*

getInfectionRate(pools,interval, target_year, target_disease,pt_estimate, species_list = c(NULL), trap_list = c(NULL))

*Arguments*

- pools: Pools data retrieved from getPools(…)
- interval: Calculation interval for abundance, accepts "collection_date","Biweek","Week", and "Month.
- target_year: Year to calculate infection rate for. This year must be present in the data.
- target_disease: The disease to calculate infection rate for–i.e. "WNV". Disease acronyms are the accepted input. To see a list of disease acronyms, run unique(pools$target_acronym).
- pt_estimate: The estimation type for infection rate. Options include: "mle","bc-"mle", "mir."
- species_list: Species filter for calculating abundance. Species_display_name is the accepted notation. To see a list of species present in your data run unique(pools$species_display_name). If species is unspecified, the default NULL will return data for all species in data.
- trap_list: Trap filter for calculating abundance. Trap_acronym is the is the accepted notation. Run unique(pools$trap_acronym) to see trap types present in your data. If trap_list is unspecified, the default NULL will return data for all trap types.

```
IR = getInfectionRate(pools,
                      interval = "Week",
                      target_year = 2023,
                      target_disease = "WNV",
                      pt_estimate = "mle",
                      species_list = c("Cx pipiens"),
                      trap_list = c("CO2","GRVD") )
IR
```

```
##    Year Week Disease Point_Estimate    Lower_CI   Upper_CI
## 1  2023   20     WNV      0.0000000  0.00000000   4.617179
## 2  2023   21     WNV      0.0000000  0.00000000   4.119261
## 3  2023   22     WNV      0.0000000  0.00000000   3.156551
## 4  2023   23     WNV      0.5727378  0.03289081   2.738134
## 5  2023   24     WNV      0.0000000  0.00000000   1.900988
## 6  2023   25     WNV      0.5406875  0.03100243   2.592643
## 7  2023   26     WNV      3.3384906  1.81130445   5.649144
## 8  2023   27     WNV      5.3460190  2.81661597   9.226036
## 9  2023   28     WNV      5.7276495  3.51402356   8.807876
## 10 2023   29     WNV      9.0820939  6.11084341  12.962783
## 11 2023   30     WNV     12.3177249  7.52367189  18.962379
## 12 2023   31     WNV      9.9889724  6.38226973  14.847046
## 13 2023   32     WNV     13.7873398  8.03997168  21.993966
## 14 2023   33     WNV     13.1298346  8.13148966  19.943141
## 15 2023   34     WNV     16.5035095 10.75997098  24.094888
## 16 2023   35     WNV      9.7385505  4.78481110  17.613511
## 17 2023   36     WNV      7.5301685  3.84942575  13.271657
## 18 2023   37     WNV      3.5282350  1.31243740   7.706321
## 19 2023   38     WNV      2.7297894  0.72200878   7.245097
## 20 2023   39     WNV      0.0000000  0.00000000   2.223982
## 21 2023   40     WNV      1.8502043  0.33078143   5.950046
## 22 2023   41     WNV      1.0256410  0.05906777   4.851477
## 23 2023   42     WNV      0.0000000  0.00000000  11.530964
## 24 2023   43     WNV      0.0000000  0.00000000  29.621925
## 25 2023   44     WNV      0.0000000  0.00000000  61.201662
## 26 2023   45     WNV      0.0000000  0.00000000  33.153285
## 27 2023   46     WNV      0.0000000  0.00000000  96.436220
## 28 2023   47     WNV      0.0000000  0.00000000  94.474839
## 29 2023   48     WNV      0.0000000  0.00000000 306.219604
## 30 2023   49     WNV      0.0000000  0.00000000 793.450686
```

## Vector Index

### getVectorIndex()

*Description*

getVectorIndex()(..) requires at least five years prior to the target_year of arthro collections data to calculate for the specified parameters. The function uses the methods of the Gateway Abundance Anomaly calculator, and will not work if there is fewer than five years of data present.

*Usage*

getVectorIndex(collections, pools, interval, target_year, target_disease,pt_estimate, species_list=NULL,trap_list = NULL)

*Arguments* - collections: collections data retrieved from getCollections(…) - pools: Pools data retrieved from getPools(…)

**Note: Years from pools and collections data must match**

- interval: Calculation interval for abundance, accepts "collection_date","Biweek","Week", and "Month.
- target_year: Year to calculate infection rate for. This year must be present in the data.

- target_disease: The disease to calculate infection rate. Disease acronyms are the accepted input. To see a list of disease acronyms, run unique(pools$target_acronym).
- pt_estimate: The estimation type for infection rate. Options include: "mle","bc-"mle", "mir."
- species_list: Species filter for calculating abundance. Species_display_name is the accepted notation. To see a list of species present in your data run unique(pools$species_display_name). If species is unspecified, the default NULL will return data for all species in data.
- trap_list: Trap filter for calculating abundance. Trap_acronym is the is the accepted notation. Run unique(pools$trap_acronym) to see trap types present in your data. If trap_list is unspecified, the default NULL will return data for all trap types.

```r
pools=getPools(token,2023,2023)
collections= getArthroCollections(token,2023,2023)

getVectorIndex(collections, pools, interval = "Biweek",
                           2023,
                           target_disease = "WNV", pt_estimate = "bc-mle",
                           species_list=c("Cx tarsalis"),

                           trap_list =  c("CO2"))
```

```
##    Biweek EPIYEAR Count Trap_Events Abundance Year Disease Point_Estimate
## 1      10    2023   509          65      7.83 2023     WNV      0.0000000
## 2      11    2023  1910         142     13.45 2023     WNV      0.5205304
## 3      12    2023  2343         153     15.31 2023     WNV      0.4359300
## 4      13    2023 12226         182     67.18 2023     WNV      0.9791292
## 5      14    2023 21573         226     95.46 2023     WNV      3.2518724
## 6      15    2023 20979         217     96.68 2023     WNV      7.0073515
## 7      16    2023 17215         255     67.51 2023     WNV     10.6965337
## 8      17    2023 11019         226     48.76 2023     WNV     10.0743079
## 9      18    2023  8184         213     38.42 2023     WNV      6.1491461
## 10     19    2023  4625         211     21.92 2023     WNV      4.5315857
## 11     20    2023  3213         211     15.23 2023     WNV      0.7800335
## 12     21    2023  1486         144     10.32 2023     WNV      0.0000000
## 13     22    2023   105          74      1.42 2023     WNV      0.0000000
## 14     23    2023    15          43      0.35 2023     WNV      0.0000000
## 15     24    2023     5          33      0.15 2023     WNV      0.0000000
##      Lower_CI   Upper_CI VectorIndex
## 1  0.00000000   6.735594    0.000000
## 2  0.02996660   2.523691    7.001133
## 3  0.02501275   2.117993    6.674089
## 4  0.43252140   1.931931   65.777899
## 5  2.37760030   4.355111  310.423742
## 6  5.56549501   8.731092  677.470744
## 7  8.67025058  13.093728  722.122990
## 8  7.95346287  12.631808  491.223255
## 9  4.45687351   8.306496  236.250194
## 10 2.78179796   7.029671   99.332359
## 11 0.13964422   2.553421   11.879910
## 12 0.00000000   3.006583    0.000000
## 13 0.00000000  29.700567    0.000000
## 14 0.00000000 213.153871    0.000000
## 15 0.00000000 367.003169    0.000000
```

# Tables

### getPoolsComparisionTable()

*Description*

getPoolsComparisionTable() produces a frequency table for positive and negative pools counts by year and species. The more years present in the data, the larger the table.

*Usage*

getPoolsComparisionTable(pools,target_disease, species_seperate=F)

*Arguments*

- pools: Pools data retrieved from getPools(...)
- target_disease: The disease to calculate infection rate for–i.e. "WNV". Disease acronyms are the accepted input. To see a list of disease acronyms, run unique(pools$target_acronym).
- species_seperate: Should the pools comparison be split by species of each pool. Default is FALSE.

```
getPoolsComparisionTable(pools, target_disease ="WNV", species_seperate = T)
```

```
## # A tibble: 62 x 7
## # Groups:   surv_year, species_display_name, Week [62]
##    surv_year  Week species_display_name Negative Confirmed Total PercentPositive
##        <int> <dbl> <chr>                   <int>     <int> <int>           <dbl>
##  1      2023    36 An freeborni                1         0     1           0
##  2      2023    46 An freeborni                2         0     2           0
##  3      2023    20 Cx pipiens                 95         0    95           0
##  4      2023    21 Cx pipiens                112         0   112           0
##  5      2023    22 Cx pipiens                140         0   140           0
##  6      2023    23 Cx pipiens                157         1   158           0.633
##  7      2023    24 Cx pipiens                150         2   152           1.32
##  8      2023    25 Cx pipiens                214         3   217           1.38
##  9      2023    26 Cx pipiens                242        17   259           6.56
## 10      2023    27 Cx pipiens                205        12   217           5.53
## # i 52 more rows
```

## Styling Dataframes with kable

Professional looking tables can be produced using the kable and kableExtra packages.

```
AbAnOutput = getAbundance(collections,
                     interval = "Biweek",

                     species_list = c("Cx tarsalis", "Cx pipiens"),
                     trap_list = "CO2",
                     species_seperate = FALSE)
head(AbAnOutput)
```

```
##   EPIYEAR Biweek Count Trap_Events Abundance
## 1    2023     10   882          65     13.57
## 2    2023     11  3254         142     22.92
## 3    2023     12  4395         153     28.73
```

| Disease Year | Biweek | Count | Trap Events | Abundance |
|---|---|---|---|---|
| 2023 | 10 | 882 | 65 | 13.57 |
| 2023 | 11 | 3254 | 142 | 22.92 |
| 2023 | 12 | 4395 | 153 | 28.73 |
| 2023 | 13 | 15803 | 182 | 86.83 |
| 2023 | 14 | 24939 | 226 | 110.35 |
| 2023 | 15 | 24113 | 217 | 111.12 |
| 2023 | 16 | 19062 | 255 | 74.75 |
| 2023 | 17 | 12865 | 226 | 56.92 |
| 2023 | 18 | 10088 | 213 | 47.36 |
| 2023 | 19 | 7161 | 211 | 33.94 |
| 2023 | 20 | 5934 | 211 | 28.12 |
| 2023 | 21 | 2806 | 144 | 19.49 |
| 2023 | 22 | 279 | 74 | 3.77 |
| 2023 | 23 | 122 | 43 | 2.84 |
| 2023 | 24 | 29 | 33 | 0.88 |

Table X: Combined biweekly Abundance Calculation for Cx. tarsalis, pipiens in CO2 traps

```
## 4    2023      13 15803         182      86.83
## 5    2023      14 24939         226     110.35
## 6    2023      15 24113         217     111.12
```

```
#As a kable table where column names, font_size, type and much more can be customized

AbAnOutput %>%
  kbl(col.names = c("Disease Year", "Biweek", "Count","Trap Events", "Abundance")) %>%
  kable_styling(bootstrap_options = "striped",
                font_size = 14,
                latex_options="scale_down")%>%
  footnote(general = "Table X: Combined biweekly Abundance Calculation for Cx. tarsalis, pipiens in CO2
```

### Data using datatables

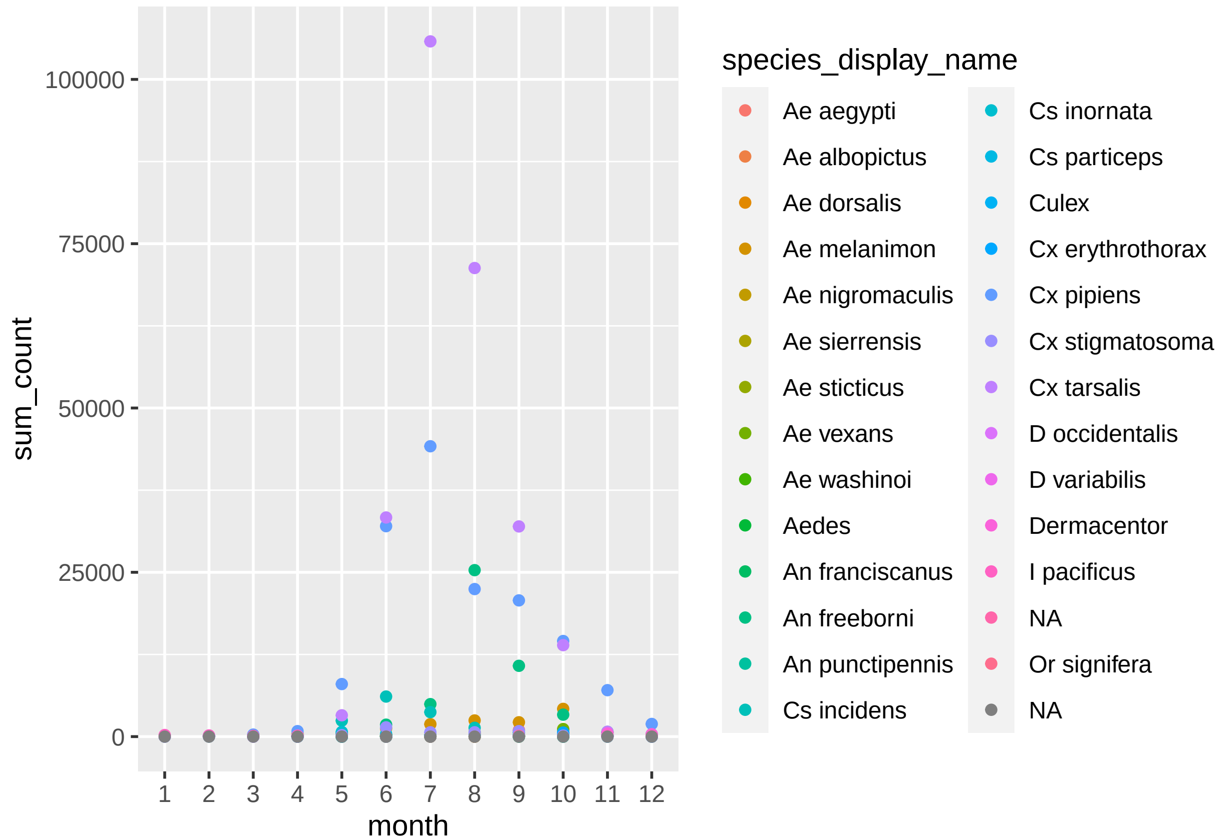
Interactive html only tables can be produced using the DT package. DT tables allow for sorting and filtering with in a webpage. These are ideal for viewing data but are not compatable with pdf or word formats.

```
#AbAnOutput %>%
  #datatable(colnames =  c("Disease Year", "Biweek", "Count","Trap Events", "Abundance"))
```

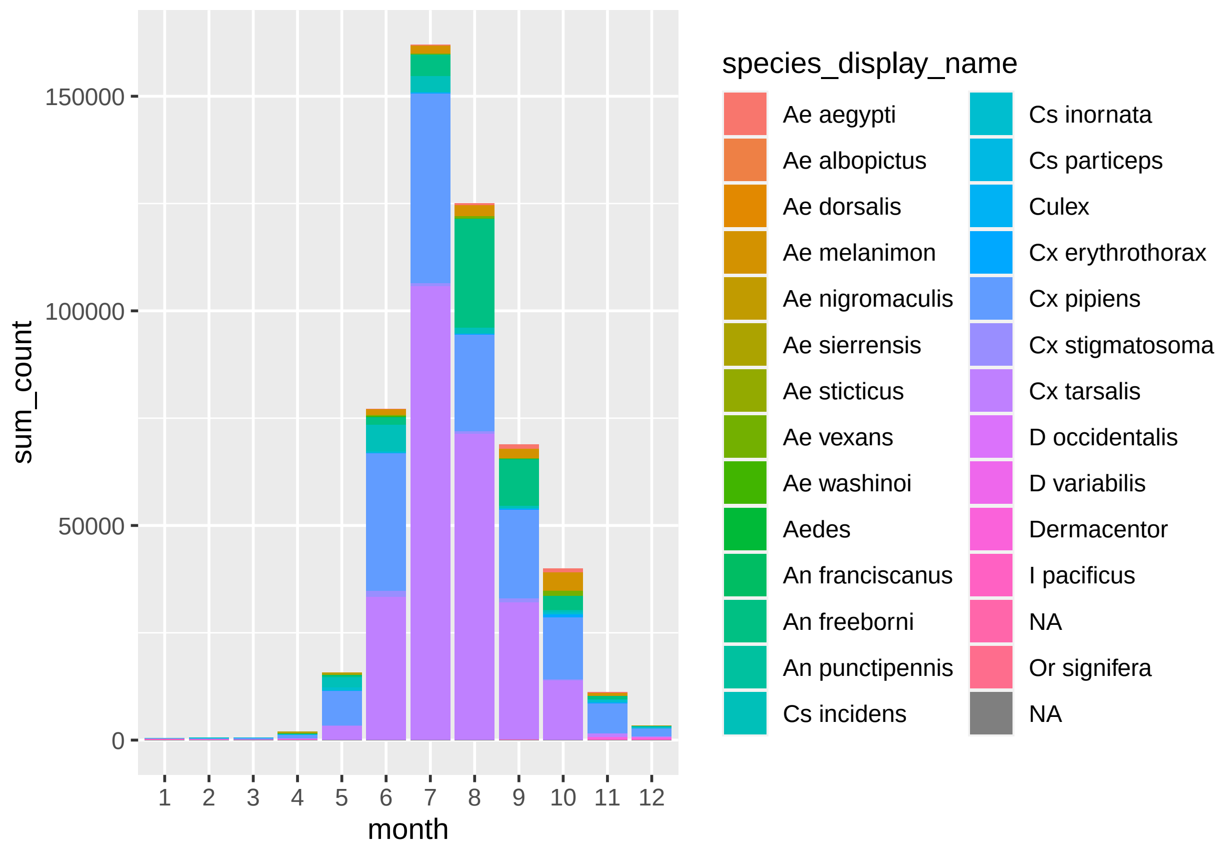
## Charts and Graphs

Ggplot is a easy to use plotting library in R. Ggplot syntax consists of creating a ggplot object with a dataframe and adding subsequent arguments to that object. Aesthetics (aes) in ggplot represents the data mapping aspect of the plot. A simple example using collections is shown below.

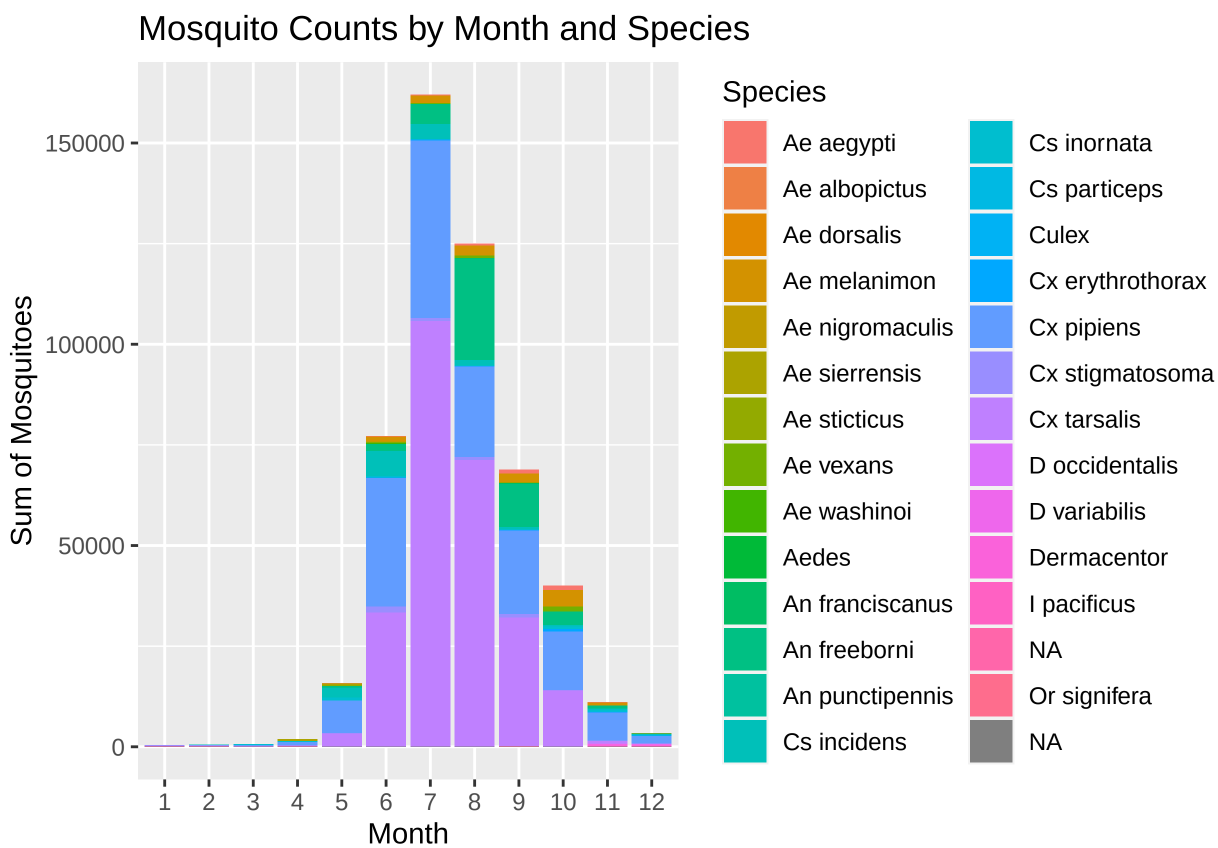
```r
#creates a month column and translates numerics
collections$month = as.factor(month(collections$collection_date))
collections_sums = collections %>%
  group_by(month, species_display_name) %>%
  summarise(sum_count = sum(num_count, na.rm=T))

#ggplot with dots a values for each species
ggplot(data=collections_sums,
       aes(x = month, y = sum_count, color = species_display_name))+
  geom_point()
```




```r
#bar chart
ggplot(data=collections_sums,
       aes(x = month, y = sum_count, fill = species_display_name))+
  geom_bar(stat="identity")
```




```r
#adding labels
ggplot(data=collections_sums,
       aes(x = month, y = sum_count, fill = species_display_name))+
  geom_bar(stat="identity") +
  labs(title = "Mosquito Counts by Month and Species",x = "Month",
       y = "Sum of Mosquitoes",
       fill = "Species")
```




When plotting with libraries in R, it is easiest when the data is prepared in long form. Most calculator outputs from our functions are in wide form. The following wrapper functions help process and plot this data.

**ProcessAbunAnom()**

*Description*

ProcessAbunAnom() processes the output returned from getAbundanceAnomaly() into a long form suitable for plotting in ggplot.

*Usage*

ProcessAbunAnom(AbAnomOutput)

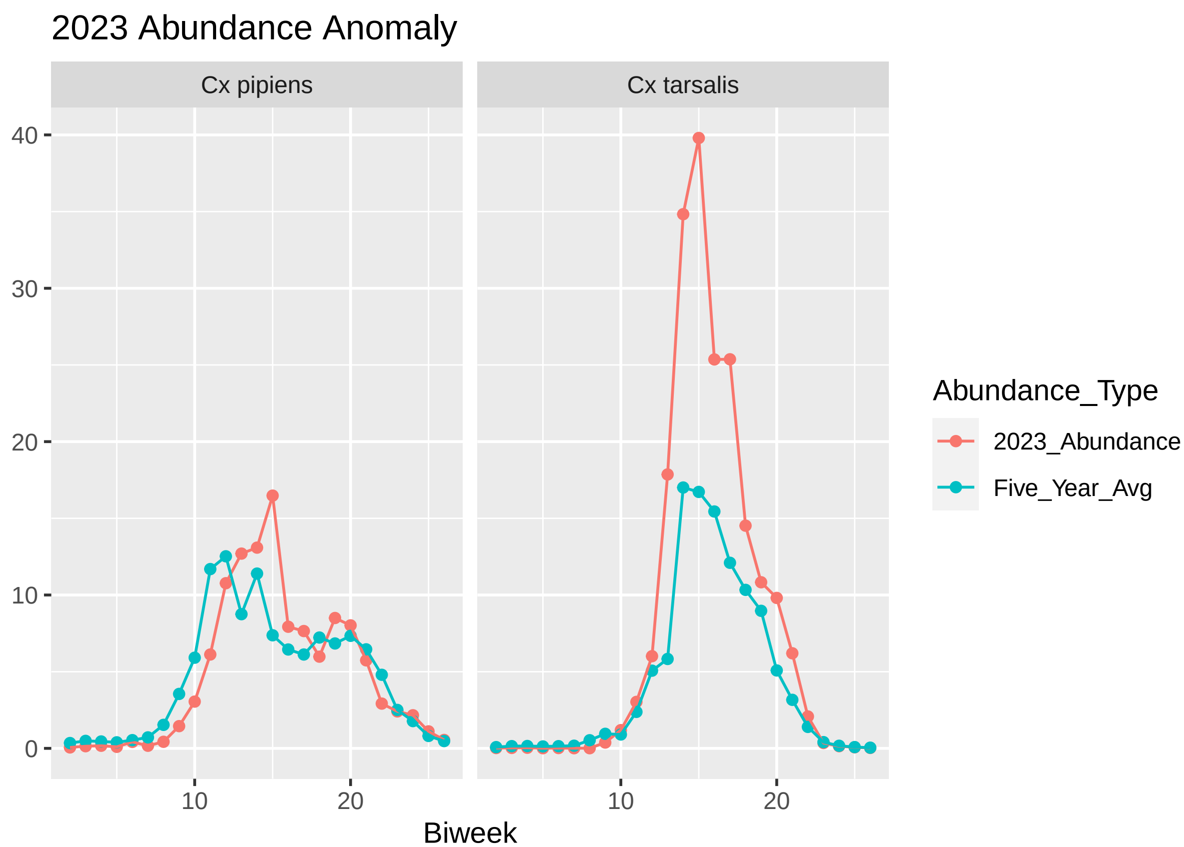
*Arguments*

- AbAnomOutput: Output from returned getAbundanceAnomaly()

```
collections= getArthroCollections(token,2018,2023)

AbAnOut = getAbundanceAnomaly(collections,
                              interval = "Biweek",
                              target_year = 2023,
                              species_list = c("Cx tarsalis", "Cx pipiens"),
                              species_seperate = TRUE) #species_seperate set to true will allow for us


AbAnOut_L = ProcessAbunAnom(AbAnOut)
```

We can take the output of ProcessAbunAnom and create a plot comparing the target year abundance to the five year average.

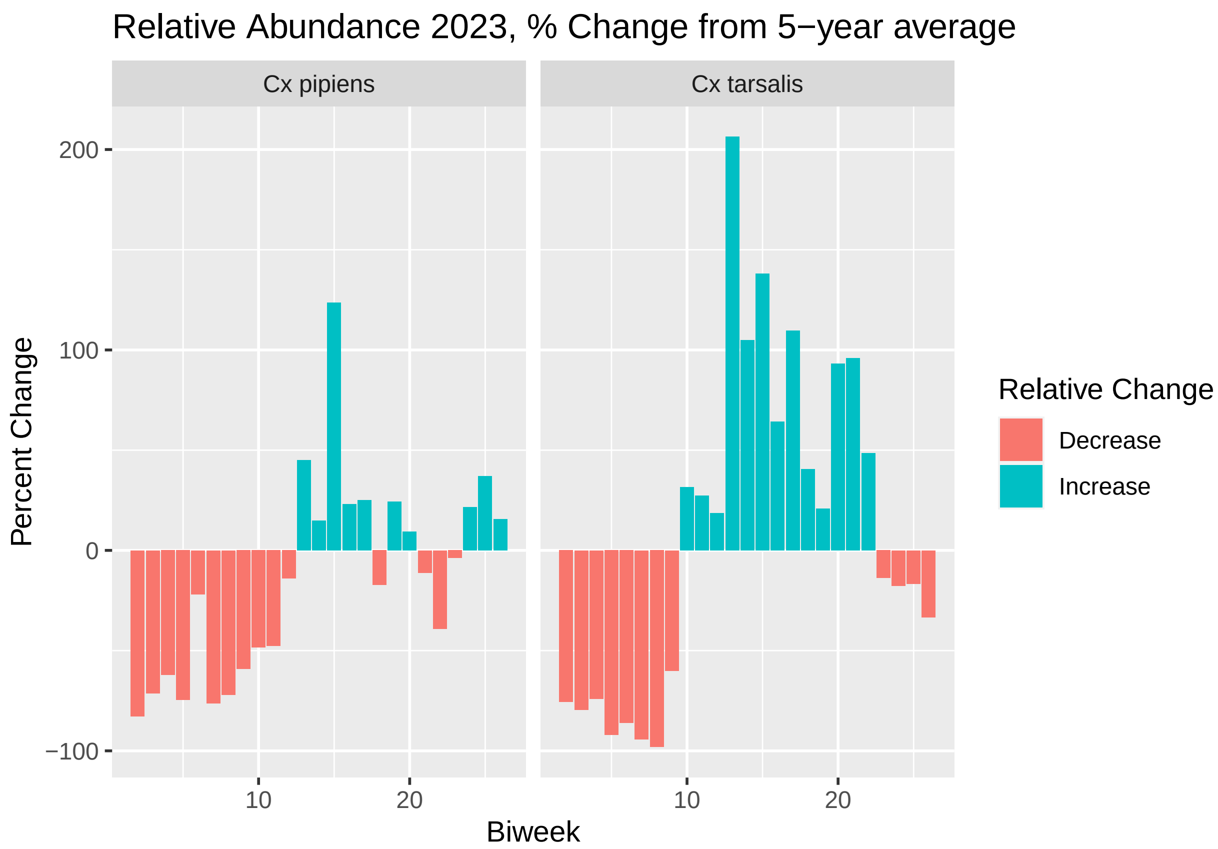
```
AbAnOut_L %>%  filter(Abundance_Type %in% c("2023_Abundance",
                                             "Five_Year_Avg"))%>%
  ggplot(aes(x=Biweek,
             y= Abundance_Calculation,
             color = Abundance_Type)) +
  geom_point()+
  geom_line() +
  facet_wrap(~species_display_name)+
  labs(title="2023 Abundance Anomaly",y = "")
```




We can also create a plot which displays the percent change from the five year average.

```
AbAnOut_L %>%
  filter(Abundance_Type =="Delta")%>%
  mutate(Change = ifelse(Abundance_Calculation > 0, "Increase", "Decrease")) %>%
  ggplot(aes(x=Biweek,
             y= Abundance_Calculation,
             fill=Change)) +
  geom_bar(stat="identity")+
  facet_wrap(~species_display_name)+
  labs(x="Biweek",
       y="Percent Change",
```

```
       title = "Relative Abundance 2023, % Change from 5-year average",
       fill="Relative Change")
```




**plotInfectionRate()**

*Description*

plotInfectionRate() plots the output returned from getInfectionRate() with confidence intervals using ggplot

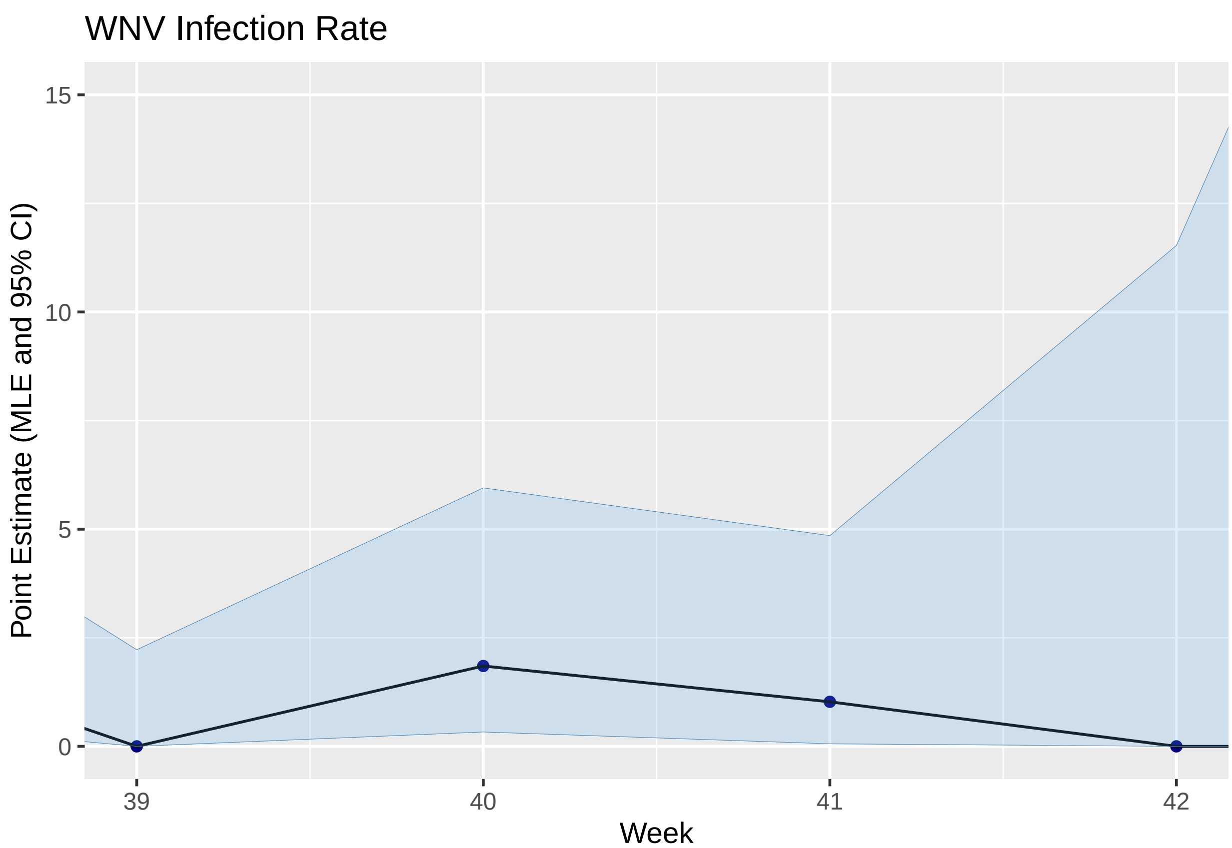
*Usage*

plotInfectionRate(InfRtOutput)

*Arguments*

- InfRtOutput: Output from returned getInfectionRate()

```
IR = getInfectionRate(pools,
                      interval = "Week",
                      target_year = 2023,
                      target_disease = "WNV",
                      pt_estimate = "mle",
                      species_list = c("Cx pipiens"),
                      trap_list = c("CO2","GRVD") )

plotInfectionRate(InfRtOutput = IR)
```




## Additional Table Examples

We can highlight rows and columns, add headers, and customize footnotes. For more information please Click Here

```
collections= getArthroCollections(token, 2021,2023)

table(collections$trap_acronym,collections$surv_year)%>%
  kbl(align = "c") %>%
    kable_paper(full_width = F,
                html_font="arial",
                lightable_options = "striped",
                ) %>%
  add_header_above(c("Trap Type", "Years" = 3)) %>%
  footnote(general = "Table X: Traps deployed by year", general_title = "") %>%
  row_spec(c(3,9,10), background = "yellow")%>%
  column_spec(c(4), background = "orange")
```

| Trap Type | Years | | |
|---|---|---|---|
| | 2021 | 2022 | 2023 |
| BACKPACK | 26 | 33 | 11 |
| BGSENT | 5600 | 7139 | 6881 |
| BTLJC | 84 | 0 | 0 |
| CO2 | 6218 | 5488 | 7431 |
| FLANNEL | 301 | 296 | 293 |
| GRVD | 8270 | 7700 | 8177 |
| LCKR | 3707 | 3693 | 3559 |
| OTHER | 10 | 11 | 37 |
| OVI | 0 | 294 | 0 |
| WRKR | 124 | 0 | 0 |

Table X: Traps deployed by year