Relevant Research Question
What is the influence of invasive plants on grassland small mammal communities?

Study design: 14 small mammal live trapping grids, 7 within native vegetation, 7 within invasive vegetation (primarily cheatgrass (*Bromus tectorum*)).
Vegetation surveys (line-point intercept) to characterize percent cover (for 27 different cover types) for each trapping grid.

Predictor variables: vegetation, seed abundance, soil, grazing intensity, weather/climate

Response variables: small mammal abundance/density, productivity, survival, community richness, habitat selection

Study Area: Thunder Basin National Grassland, WY (northeast)

Vegetation Data
Line-point intercept surveys were conducted on all small mammal trapping grids to estimate percent cover for different cover types. There are 14 trapping grids and 8 transects were surveyed per grid, totaling 112 transects. Although most plants were not identified to species, there are still 27 different cover types in the dataset. See section 1.c in the script for a list of cover types. We surveyed 50 points per transect, thus percent cover is calculated by counting the number of points where a certain cover type is present, then multiplying that count by two. If we had surveyed 25 or 100 points per transect we would multiply the count by 4 or 1, respectively. This portion of the analysis is not the focus of this project, but I used the *aggregate* function and the code is in section 2 of my script.

The dataset this project uses is the output from the *aggregate* function mentioned above, which contains percent cover for each cover type, by grid and transect:

![Table](image)

Fig 1. Subset of percent cover dataset (a vertical table). Columns TL to Surf are different vertical layers for a single point. Numbers are percents.
Data Management Challenge: accounting for all cover type by transect + grid combinations
(adding zeros to the dataset)

Data was only recorded when a cover type was present. Thus, the Percent Cover (PC) dataset only contains rows where at least one vertical layer has a non-zero value. In other words, if type “AC” was not found on transect V1 on grid 1-IE then that information is absent from the dataset. The data management challenge is thus to add all missing zeros to the dataset so, for example, mean PC can be calculated for each cover type.

As stated previously, there are 14 grids and 8 transects were surveyed per grid, totaling 112 transects. Additionally, there are 27 different cover types. Thus, the PC dataset with zeros added to it should have 112*27 rows = 3024. Then there would be data for each cover type by transect plus grid combination. The initial PC dataset has 820 rows, which means there are 3024-820 = 2204 missing zeros (or rows).

Setdiff, Expand.grid, and Interaction Functions

I used the setdiff function to find every transect + grid combination that was missing for a given cover type (see section 1.b). Setdiff returns a list of values that are not matched between the two variables being compared. For example, if A = c(1,2,3,4,5) and B = c(4,5,6,7,8) then setdiff(B, A) will return: 6, 7, 8; whereas setdiff(A, B) will return: 1,2,3. So, obviously order matters. In order to use setdiff, I created a variable (factor) with all the possible transect + grid combinations, called “gv” for “grouping variable”. Since I needed to group by two variables, I combined transect and grid into a single term using, 1) expand.grid function, and 2) interaction function (see section 1.a). Expand.grid created a new dataframe with 2 columns, where each row is a unique combination of transect and grid. Interaction converted the 2 columns into a factor where each level is, again, a unique combination of transect and grid. For example, the data went from transect V1 and grid 1-IE in two separate columns to V1.1-IE as a single term (or level) in a factor. Finally the setdiff function compared “gv” to the dataframe subsetted to a certain cover type to assess mismatches (see section 1.b).

Loop and Merge

After testing setdiff on a subset of the dataframe, I used a “for loop” to find every transect + grid combination that was missing for each cover type. I looped cover type to create a subset of the dataframe within the loop and then used the setdiff function to find the missing values (see section 1.c). The loop returns a dataframe containing the grouping variable and cover type for all missing values. I then used merge to combine the dataframe from the loop with the original PC dataframe (see section 1.d). An important default setting for merge is the “all=FALSE” argument, which means rows are dropped.
(omitted) that are not present in both dataframes. In my case, the rows I was appending were necessarily not present in the PC dataframe since I was looking for missing rows. Thus, I set “all = TRUE”, which keeps all rows in both dataframes.

**Substr Function**

After merging the output from the loop to the PC dataframe, the PC dataframe contained a “grouping variable” column with transect and grid as a single term. I separated transect and grid into separate columns using the `substr` function and then removed the “grouping variable” column (see section 1.e). `Substr` is a useful function that takes as arguments: 1) the object being referenced, 2) the character number where extraction starts, and 3) the character number where extraction ends. For example, if I want to only select the numbers from the string, A <- c("asd123az"), the code is: `substr(A, 4,6)`, returning “123”. `Substr` can also be used to make replacements, see “?substr”. For example, if AA <- c("asd123az"), and you want to replace “123” with “456”, the code is: `substr(A, 4, 6) <- c("456")`, which returns “asc456az”.

**Grep Function**

The `grep` function acts as a “like” argument: rather than returning values within a column that are an exact match (equal) to a term, `grep` returns values within a column that contain a certain term but also may contain other terms. For example, I used `grep` to code my grids either “I” (invasive) or “N” (native) based on values in the column GridName (see section 1.f). Grids within GridName only contain an “I” if they are invasive and only have an “N” if they are native, for example: “1-IE” and “1-NN”. I thus used `grep` to search for “I” or “N” in GridName and create a new GridCode column where “I” or “N” was assigned, allowing me to group by all native or all invasive grids.

**Find and Replace**

The final step in creating the new dataframe was replacing the remaining NA’s with zeros. After the above steps, the remaining NA’s only represent percent cover values for the rows that were identified as missing with `setdiff`. Thus, all of the remaining NA’s should be zeros, since they are locations where a certain cover type was absent. A simple line of code accomplishes the last task: `Data[is.na(Data)] <- 0` (see section 1.g). Additionally, if you only wanted to replace NA’s within a certain column: `Data$col[is.na(Data$col)] <- 0`. Or, if you needed to replace a non-NA value with a different value, for example if you needed a binary predictor or response: `Data$col[Data$col == 2] <- 0`. This code would change all 2’s within “col” into 0’s. If you are referencing characters or factors, they need to be in quotations: `Data$col[Data$col == “x”] <- “y”`. 