Extract landcover covariates

Aidan Brushett

2025-05-02

Contents

Before you begin

This script is number 1 of 6 in a series of scripts used to replicate the analyses presented in the paper: “Life on the edge: Industrial footprint and edge effects variably affect the distribution of a boreal small mammal”

This script was used to extract the landscape metrics for all sites in the study from a variety of rasters, geodatabases, and shapefiles of landcover data. **The raw spatial data used to develop landcover metrics is not available on GitHub but can be shared by the authors upon request.**

When running these scripts, please ensure that you have downloaded the complete GitHub repository. This will ensure you have all the files, data, and proper folder structure you will need to run this code and associated analyses.

Also make sure you open RStudio through the R project (OSM_red_squirrel_distribution.Rproj). This will automatically set your working directory to the correct place (wherever you saved the repository) and ensure you don't have to change the file paths for some of the data. This analysis was initially run in R v4.3.0. If you have any questions or concerns, please contact one of the authors (in order):

Aidan Brushett M.Sc. Student University of Victoria
School of Environmental Studies
Email: aidanbrushett@uvic.ca

Emerald Arthurs M.Sc. Student University of Victoria
School of Environmental Studies

---

0. Setup

Don't want to run anything in knitting because it would take a bazillion hours. Everything set to `eval = FALSE` for now.

```
library(sf) # for points and buffering
library(tidyverse)
rm(list = ls())
```

1. Forest inventory data

1.1. Load, clip the forest inventory data

```
# Camera locations (point)
sites <- st_read("./maps/OSM_mapping.gdb", layer = "all_arrays_locations_covariates") %>%

  filter(str_detect(array, "LU")) %>%

  select(array, site)

plot(sites[1])

# buffer sites so that we can filter the landcover data and reduce file size
sites_buffered <- st_buffer(sites, dist = 10000) %>%

  st_union()

# Inspect the result
plot(sites_buffered[1])
```

```
# load landcover layer (will be slow)
landcover_unclipped <- st_read("./maps/SBFI_2020_Boreal_Plains.gdb", layer = "SBFI_2020_polygons")

landcover <- landcover_unclipped %>%

  st_transform(., crs = 26912) %>%

  filter(lengths(st_intersects(., sites_buffered)) > 0)

#make sure it's all projected correctly
st_crs(landcover, describe = TRUE)
st_crs(sites, describe = TRUE)
plot(landcover[1])

# clean up
rm(landcover_unclipped, sites_buffered)
gc()

# double check that they are all format "sf" and "dataframe"
str(sites)
str(landcover)
```

1.2. Reshape data for key variables

Tree species composition:

```
# Tree species present in the dataset
tree_spp <- landcover %>%

  st_drop_geometry() %>%

  select(SPECIES_1,
         SPECIES_2,
         SPECIES_3,
         SPECIES_4,
         SPECIES_5) %>%

  unlist(.) %>%   # In a big vector

  unique(.) %>%

  .[ . != " "] # Remove empty strings

# Re-pivot the tree data for each polygon. This can take a while and is a bit clunky.
tree_percent <- landcover %>%

  st_drop_geometry() %>%

  # Select the species columns that have the name and % comp only
  select(ID,
         contains("SPECIES") &
           !contains("_CML"),
         -SPECIES_NUMBER,
         -SPECIES_CONIFEROUS_PERC
         ) %>%

  mutate(across(everything(), as.character)) %>%  # Convert all columns to character
```

```
  # The start of some funky data reshaping
  pivot_longer(cols = starts_with("SPECIES_"),
               names_to = "temp",
               values_to = "val") %>%

  # What rank is the species? We don't actually care but this helps keep things tidy/avoids
  ↪  duplicates when we pivot wider later
  mutate(spp_rank = as.numeric(str_extract(temp, "\\d+"))) %>%

  # Remove the numbers so that all the former rank columns are just called "species"
  mutate(temp = str_replace_all(temp, "_\\d+", ""),
         temp = str_replace_all(temp, "SPECIES_PERC", "PERCENTAGE")) %>%

  # Pivot wider so we now have a species column and a percent column
  pivot_wider(names_from = temp, values_from = val) %>%

  # Only want the non-zero percents
  filter(PERCENTAGE > 0) %>%

  # Pivot wider again to get our species and comp values!!
  pivot_wider(names_from = SPECIES, values_from = PERCENTAGE, values_fill = "") %>%

  select(-spp_rank) %>%

  mutate(across(everything(), as.numeric)) %>%

  # Replace NAs
  mutate(across(everything(), ~ replace(., is.na(.), 0))) %>%

  group_by(ID) %>%

  # Take the sum of the % comp for all the species
  summarize(across(everything(), sum))
```

Percent harvest by year:

```
# Get harvest percent
harvest_percent <- landcover %>%

  # Runs faster without coordinates
  st_drop_geometry(.) %>%

  # Year and amount columns, plus ID for pivoting/joining later
  select(ID,
         contains("DISTURB_HARVEST")) %>%

  # Reshape to a column for each year
  pivot_wider(names_from = DISTURB_HARVEST_YEAR,
              values_from = DISTURB_HARVEST_PERC,
              values_fill = 0) %>%

  select(-`0`) %>%

  # clean up the names
  rename_with(., ~paste0("HARVEST_PCT_", .), .cols = !contains("ID")) %>%

  # order by year
  select(ID, sort(names(.)))
```

Percent burn by year:

```
# Get fire percent
fire_percent <- landcover %>%

  st_drop_geometry(.) %>%

  # Year and proportion columns, plus ID for pivoting/joining later
  select(ID,
         DISTURB_FIRE_YEAR,
         DISTURB_FIRE_PERC) %>%

  # Reshape to a column for each year
  pivot_wider(names_from = DISTURB_FIRE_YEAR,
              values_from = DISTURB_FIRE_PERC,
              values_fill = 0) %>%

  select(-`0`) %>%

  # clean up the names
  rename_with(., ~paste0("FIRE_PCT_", .), .cols = !contains("ID")) %>%

  select(ID, sort(names(.)))
```

Intensity of fire for a given polygon. Same code as above just with a new variable of interest.

```
fire_intensity <- landcover %>%

  st_drop_geometry(.) %>%

  select(ID,
         DISTURB_FIRE_MAGNITUDE_AVG,
         DISTURB_FIRE_YEAR) %>%

  pivot_wider(names_from = DISTURB_FIRE_YEAR,
              values_from = DISTURB_FIRE_MAGNITUDE_AVG,
              values_fill = 0) %>%

  select(-`0`) %>%

  rename_with(., ~paste0("FIRE_MAG_", .), .cols = !contains("ID")) %>%

  select(ID, sort(names(.)))
```

Left join all of the new covariates to the original spatial landcover data and replace NAs induced by joining rows with no fire/harvest.

```
landcover_recalculated <- landcover %>%

  left_join(tree_percent, by = "ID") %>%

  left_join(fire_percent, by = "ID") %>%

  left_join(fire_intensity, by = "ID") %>%
```

```
  left_join(harvest_percent, by = "ID") %>%

  # Replace NAs
  mutate(across(
        .cols = c(all_of(tree_spp), contains("FIRE_PCT"), contains("HARVEST_PCT")),
        ~ replace(., is.na(.), 0))) %>%

  # Original tree species names were ugly, should fix later.
  rename_with(., ~paste0(., "_PCT_OF_TREED"), .cols = all_of(tree_spp))

rm(tree_spp, tree_percent, fire_percent, fire_intensity, harvest_percent)
```

1.3. Identify variables to extract

```
landcover_metrics <- landcover_recalculated %>%

  # Select the metrics that you want to calculate
  select(.,

         # Forest age
         contains("AGE_"),
         -AGE_AVG,
         -AGE_MIN,
         -AGE_MAX,
         -AGE_MEDIAN,
         -AGE_SD,

         # Landcover
         contains("LC_"),
         -LC_FAO_FOREST,
         -LC_TREED,
         -LC_WETLAND_VEGETATION,

         # Tree composition
         contains("_PCT_OF_TREED"),

         # Fire and harvest percent
         contains("FIRE_PCT"),
         contains("HARVEST_PCT")
         )
```

1.4. Extract data for the sites of interest

This sums the % cover within each polygon and is weighted by their proportion of area within the buffer.

- e.g. a polygon takes up 0.50 of the buffer and was 50% conifer = 25% conifer total
- an additional polygon takes up 0.25 of the buffer and was 80% conifer = 20% conifer total
- the proportion cover for the total buffer around the site is 45% conifer.
- *within a polygon the vegetation attributes are assumed to be constant*

```
sites_landcover_metrics <- list() # To store results
```

```r
# For progress bar
start_time <- Sys.time()

# A `for` loop is nicer for progress statements IMO vs. purrr...
for(i in 1:nrow(sites)) {

  progress <- (i - 1) / nrow(sites)

  # Site we are extracting
  site <- sites[i,]

  # Progress bar
  cat("\r Extracting site ", i, " of ", nrow(sites), " (", site$site, "): ",
      round(as.numeric(difftime(Sys.time(), start_time, units = "hours")) * (1 - progress) /
       ↪  progress, 2),
      " hours remaining        ", sep = "")

  # Store results in a list
  sites_landcover_metrics[[i]] <-

    # Buffer distances for which we are extracting using purrr
    purrr::map_dfr(c(50, 100, seq(250, 5000, by = 250)), ~

      # Clip the landcover to the site buffer
      st_intersection(landcover_metrics,
                      st_buffer(site, .x)) %>%

      mutate(feature_area = st_area(.) %>% # Area of the feature within the buffer
               as.numeric(.),

             # Total area sampled. For the LANDCOVER ONLY DATA, we will remove NA areas from our
              ↪  'total' area calculation.
             # Obviously this doesn't make sense for HFI data.
             total_area = sum(feature_area, na.rm = TRUE) %>%
               as.numeric(.),

             # Percent cover by a given feature
             percent_area = feature_area / total_area,

             # Values for 'proportion cover' adjusted by the amount of buffer area (see above Rmd
              ↪  notes)
             across(where(is.numeric) & !contains("_area"), ~ . / 100 * percent_area)) %>%

      # Runs faster without geometry
      st_drop_geometry(.) %>%

      # Sum across all features
      summarise(across(where(is.numeric) & !contains("_area"), sum, na.rm = TRUE)) %>%

      # Keep track of where the numbers came from!
      mutate(buffer_dist = .x,
             array = site$array,
             site = site$site)

    )

  # Clear some memory
  rm(site)
  gc()
```

```
}

# Save a backup in the same chunk
save(sites_landcover_metrics, file = "./data/raw/OSM_SBFI2020_metrics.RData")
```

1.5. Save results

```
sites_landcover_metrics_df <- bind_rows(sites_landcover_metrics) %>%

  relocate(array, site, buffer_dist)

write_csv(sites_landcover_metrics_df, "./data/raw/OSM_SBFI2020_metrics.csv")
```

---

2. Human footprint data

2.1. Import data

```
# Camera locations (point)
sites <- st_read("./maps/OSM_mapping.gdb", layer = "all_arrays_locations_covariates") %>%

  filter(str_detect(array, "LU")) %>%

  select(array, site)

plot(sites[1])

# buffer sites so that we can filter the landcover data and reduce file size
sites_buffered <- st_buffer(sites, dist = 10000) %>%

  st_union()

plot(sites_buffered[1])

# Human footprint index:
# load human features layer (will be slow)
st_layers("./maps/HFI2021_clipped.gdb")

hfi <- st_read("./maps/HFI_clipped/HFI_2021_clipped.shp") %>%

  st_transform(., crs = 26912)


#make sure it's all projected correctly
st_crs(hfi, describe = TRUE)
st_crs(sites, describe = TRUE)
plot(hfi[1])
```

```r
# clean up
rm(sites_buffered)
gc()

# double check that they are all format "sf" and "dataframe"
str(sites)
str(hfi)
```

2.2. Extract variables

```r
# Extract HFI (Human Footprint Inventory) covariates using site buffers
hfi_metrics <- hfi %>%
  select(OBJECTID, FEATURE_TY, YEAR) # Keep only relevant columns

sites_hfi_metrics <- list() # List to store output

start_time <- Sys.time() # Start timing the process

# Loop over each site in the 'sites' object
for(i in 1:nrow(sites)) {

  progress <- (i - 1) / nrow(sites) # Track progress

  site <- sites[i, ] # Extract the current site

  # Print progress to console with estimated time remaining
  cat("\r Extracting site ", i, " of ", nrow(sites), " (", site$site, "): ",
      round(as.numeric(difftime(Sys.time(), start_time, units = "hours")) * (1 - progress) /
      ↪  progress, 2),
      " hours remaining        ", sep = "")

  # Extract metrics for the current site across buffer distances
  sites_hfi_metrics[[i]] <-

    purrr::map_dfr(c(50, 100, seq(250, 5000, by = 250)), ~{

      site_buffer <- st_buffer(site, .x) # Create buffer around site

      st_intersection(hfi_metrics, site_buffer) %>% # Intersect with HFI layer

        # Calculate areas and percent cover
        mutate(feature_area = st_area(.) %>%
                 as.numeric(.), # Area of each intersected HFI polygon

               total_area = st_area(site_buffer) %>%
                 as.numeric(.), # Total area of buffer (unlike landcover, we use full area)

               percent_area = feature_area / total_area) %>% # Proportion of buffer occupied

        st_drop_geometry(.) %>% # Drop geometry for efficiency

        # Differentiate harvested areas by year
        mutate(FEATURE_TY = ifelse(FEATURE_TY == "HARVEST-AREA",
                                   paste0(FEATURE_TY, "-", YEAR),
                                   FEATURE_TY)) %>%
```

```
        # Group by array/site/feature type to summarize percent cover
        group_by(array, site, FEATURE_TY) %>%

        summarize(PCT_COVER = sum(percent_area),
                  buffer_dist = .x) %>%

        # Pivot to wide format: one row per site-buffer combo, features as columns
        pivot_wider(names_from = FEATURE_TY,
                    values_from = PCT_COVER,
                    values_fill = 0)
    })

  # Clean up memory after each iteration
  rm(site)
  gc()
}

save(sites_hfi_metrics, file = "./data/raw/OSM_HFI2021_metrics.RData")
```

2.3. Save results

```
# Some extra tidying. We need to deal with missing buffers/sites when there were no human features.
↪
sites_hfi_metrics_df <- bind_rows(sites_hfi_metrics) %>%

  ungroup() %>%

  relocate(array, site, buffer_dist, sort(names(.))) %>%

  select(-array) %>%

  complete(site, buffer_dist) %>%

  # Create some extra columns
  separate_wider_delim(site,
                       delim = '_',
                       names = c('array', 'camera'),
                       cols_remove = FALSE) %>%

  select(-camera) %>%

  # Replace NAs introduced by "complete"
  mutate(across(4:last_col(), ~replace(., is.na(.), 0)))

write_csv(sites_hfi_metrics_df, "./data/raw/OSM_HFI2021_metrics.csv")
```

---

3. Configuration metrics

3.1. Load additional packages (if not done earlier)

```
library(tidyverse)
library(terra)
library(sf)
library(landscapemetrics)
```

3.2. Import the rasters

Rasters we want:

```
configuration_simple <- rast("./rasters/OSM_landcover_HFI_binary.tif")
plot(configuration_simple)

#configuration_simplified <- rast("./rasters/OSM_HFI_SBFI_forested_classes.tif")
#plot(configuration_simplified)

configuration_grouped <- rast("./rasters/OSM_HFI_SBFI_all_veg_classes.tif")
plot(configuration_grouped)
```

Classification schemes that tell us which landcover type each raster value corresponds to (I made this manually while I was processing the rasters):

```
features_grouped <- read_csv("./rasters/OSM_HFI_SBFI_raster_feature_types.csv") %>%

  # Columns that tell us what we want
  select(feature = name.OSM_HFI_SBFI_all_veg_classes, class = value.OSM_HFI_SBFI_all_veg_classes)
   ↪  %>%

  mutate(class = as.character(class)) %>%

  distinct()

#features_simplified <- read_csv("./rasters/OSM_landcover_HFI_feature_types.csv") %>%
#
#  select(feature = feature_simplified, class = value_simplified) %>%
#
#  mutate(class = as.character(class)) %>%
#
#  distinct()
#
#plot(landscape)
#check_landscape(configuration)
```

The sites for which we are going to extract covariates:

```
sites <- st_read("./maps/OSM_mapping.gdb", layer = "all_arrays_locations_covariates") %>%

  filter(str_detect(array, "LU")) %>%
```

```
  select(array, site)

plot(sites[1])
```

3.3. Extract 'simple' covariates

To do this, we will represent the landscape as a simplified raster of human and non-human landcover. We'll extract:

- **mean core area index of natural habitat** as a representation of disturbance relief (is most habitat edge or core?). Measures mean ratio of core to edge habitat in patches (CAI = 0 when the patch has no core area and approaches CAI = 100 with increasing percentage of core area within a patch.)
- **edge density of natural habitat** (as a representation of edge *created by* interfaces with human disturbances)
- **total core area of natural habitat** – amount of natural habitat that is > 50 m from edges
- **natural habitat cohesion** – a measure of the chances two points of a class are connected. Similar to mesh size but considers edge lengths too. So a perforated forest that's still one patch will go down.

```
# Create a buffer of 5500m around the site
# Convert the buffer to a SpatVector for use with terra
# Extract the raster values within the buffer

# Extract landscape configuration metrics using raster with simplified landcover classes
# Output: Class-level landscape metrics within buffers of increasing size around each site

config_data <- list() # List to store output

start_time <- Sys.time() # Start timing the process

# Loop over each site in the 'sites' object
for(i in 1:nrow(sites)){

  progress <- (i - 1) / nrow(sites) # Track progress

  site <- sites[i, ] # Extract the current site

  # Print progress to console with estimated time remaining
  cat("\r Extracting site ", i, " of ", nrow(sites), " (", site$site, "): ",
      round(as.numeric(difftime(Sys.time(), start_time, units = "hours")) * (1 - progress) /
       ↪  progress, 2),
      " hours remaining        ", sep = "")

  # Crop the configuration raster (simplified landcover) to a 5100 m buffer around the site
  # This ensures all smaller buffers are included while improving efficiency
  config_local <- crop(configuration_simple,
                       vect(st_buffer(site, dist = 5100))) # Convert site buffer to SpatVector for
                        ↪  terra compatibility

  config_data_lu <- list() # Store buffer-specific metrics for current site

  # Loop over a range of buffer sizes
  for (size in c(50, 100, seq(250, 5000, by = 250))) {

    cat("\r Extracting site", i, "of", nrow(sites), "(", site$site, "//", size,
        "m buffer ): ", round(as.numeric(difftime(Sys.time(), start_time, units = "hours")) * (1 -
         ↪  progress) / progress, 4),
```

```
        " hours remaining    ")

    # Calculate landscape metrics within the buffer using landscapemetrics::sample_lsm()
    config_data_lu[[as.character(size)]] <- sample_lsm(

      landscape = config_local,     # Raster clipped to local area
      y = site,                     # Point location to buffer around
      shape = "circle",             # Circular buffer
      size = size,                  # Current buffer radius
      what = c(                     # Metrics to calculate (class-level metrics)
        "lsm_c_ed",                 # Edge density
        "lsm_c_cai_mn",            # Mean core area index
        "lsm_c_tca",                # Total core area
        "lsm_c_cohesion"            # Cohesion
        ),
      directions = 8,               # Queen's case connectivity (8 directions)
      edge_depth = 5,               # 25 m edge width (5 pixels @ 5 m resolution)
      consider_boundary = TRUE,     # Cells bordering landscape boundary still counted as core
      all_classes = TRUE,           # Return metrics even for absent classes (with NA)
      plot_id = site$site,          # Track site ID for reference
      verbose = TRUE,               # Show internal messages
      progress = FALSE,             # Disable internal progress bar
      return_raster = FALSE         # Do not return masked raster

      ) %>%

      # Post-processing of results
      mutate(buffer = size,                              # Track buffer size
             class = as.character(class) %>%             # Convert class ID to character
               replace_na("landscape"),                  # Rename NA to "landscape"
             class = case_when(                          # Label classes for interpretability
               class == "0" ~ "water",
               class == "1" ~ "natural",
               class == "2" ~ "anthropogenic",
               TRUE ~ class)) %>%

      select(-id, -layer) # Drop unnecessary columms

    gc() # Clean up memory after each buffer run
  }

  # Combine all buffer-level metrics into a single data frame for the site
  config_data[[site$site]] <- bind_rows(config_data_lu)
}

# Save output to file as a backup
save(config_data, file = "./data/raw/OSM_simple_config_landscapemetrics.RData")
```

3.4. Tidy, format, save the results:

```
load("./data/raw/OSM_simple_config_landscapemetrics.RData")

config_simple <- config_data %>%

  bind_rows(.) %>%
```

```
  filter(!(level == 'class' & class == "LANDSCAPE")) %>%

  mutate(covariate = paste0(tolower(class), "_", metric),
         site = plot_id) %>%

  select(-metric, -level, -class, -percentage_inside, -plot_id) %>%

  pivot_wider(names_from = covariate,
              values_from = value)

write_csv(config_simple, "./data/raw/OSM_simple_config_landscapemetrics.csv")
```

3.5. Extract more complex covariates

This means all the original landcover classes, found in the 'all_veg_classes' i.e. 'grouped' landcover raster. We will extract:

- **landscape-scale number of patches** as a measure of patch size – is the landscape a few large patches or many small ones? *This will need to be standardized by area post-hoc*
- **landscape-scale mesh size** – the probability of two pixels on the landscape being in the same patch
- **landscape scale shannon evenness index** – a relative measure of dominance
- **landscape scale simpson evenness index** – a relative measure of dominance that is less sensitive to rare species

```
# Extract landscape-level configuration metrics using raster with grouped OSM landcover classes
# Output: Landscape metrics within buffers of increasing size around each site

config_data <- list() # List to store results for each site

start_time <- Sys.time() # Start timing for progress estimates

# Loop over each site
for(i in 1:nrow(sites)) {

  progress <- (i - 1) / nrow(sites) # Track progress across sites

  site <- sites[i, ] # Extract the current site

  # Print progress and estimated time remaining
  cat("\r Extracting site ", i, " of ", nrow(sites), " (", site$site, "): ",
      round(as.numeric(difftime(Sys.time(), start_time, units = "hours")) * (1 - progress) /
       ↪  progress, 2),
      " hours remaining        ", sep = "")

  # Clip grouped landcover raster to a 5500 m buffer around the site
  # This ensures all smaller buffers will be fully captured
  config_local <- crop(configuration_grouped,
                       vect(st_buffer(site, dist = 5500))) # Convert to SpatVector for terra
                        ↪  compatibility

  config_data_lu <- list() # Temporary list for buffer-wise results at current site

  # Loop through buffer distances
  for (size in c(50, 100, seq(250, 5000, by = 250))) {
```

```
    # Extract landscape-level metrics for this buffer size
    config_data_lu[[as.character(size)]] <- sample_lsm(

      landscape = config_local,     # Cropped raster
      y = site,                     # Center buffer on this site
      shape = "circle",             # Circular buffer
      size = size,                  # Current buffer size
      what = c(                     # Landscape-level metrics to extract
        "lsm_l_np",                 # Number of patches
        "lsm_l_cohesion",          # Patch cohesion index
        "lsm_l_mesh",              # Effective mesh size
        "lsm_l_shei",              # Shannon's evenness index
        "lsm_l_siei",              # Simpson's evenness index
        "lsm_l_contag"            # Contagion index
        # "lsm_c_enn_mn"           # (Optional) Class-level mean nearest neighbour distance
        ),
      directions = 8,               # Use Queen's case (8-cell connectivity)
      edge_depth = 5,               # 25 m edge depth (5 pixels @ 5 m resolution)
      consider_boundary = TRUE,     # Count core areas adjacent to raster edge
      all_classes = TRUE,           # Retain all class metrics even if NA
      plot_id = site$site,          # Track site ID
      verbose = TRUE,               # Show warnings
      progress = FALSE,             # Disable internal progress messages
      return_raster = FALSE         # Do not return the masked raster object

      ) %>%

      mutate(buffer = size) %>%      # Track the buffer size used
      select(-id, -layer)            # Drop unused columns

    gc() # Clean up memory
  }

  # Bind all buffer sizes into a single data frame for this site
  config_data[[site$site]] <- bind_rows(config_data_lu)

}

# Save results to an .RData file for backup
save(config_data, file = "./data/raw/OSM_grouped_config_landscapemetrics.RData")
```

3.6. Tidy, format, save the results:

```
load("./data/raw/OSM_grouped_config_landscapemetrics.RData")

config_grouped <- config_data %>%

  bind_rows(.) %>%

  # landscape scale metrics (e.g. splitting index have an NA that we want to replace)
  mutate(class = ifelse(level == "landscape", "landscape", as.character(class))) %>%

  drop_na(class) %>%

  # Add the feature names
  left_join(features_grouped, by = 'class') %>%
```

```
  # If class is NA this means it was a landscape-level metric
  mutate(class = tolower(feature) %>%
           replace_na(., "landscape")) %>%

  # Name of the metric (level + type)
  mutate(covariate = paste0(tolower(class), "_", metric),
         site = plot_id) %>%

  select(-metric, -level, -class, -percentage_inside, -plot_id, -feature) %>%

  pivot_wider(names_from = covariate,
              values_from = value)

write_csv(config_grouped, "./data/raw/OSM_grouped_config_landscapemetrics.csv")
```