Script Review: "0_EFI_CaretEnsemble_...R"

SRM-LQ

24/01/2022

Contents

Action

This is an R Markdown document showing the script run-through and code edits made during the last meeting. There was also mention of the need for better ways to edit and peer-review future script development. Some R-users have recommended using trackdown with github functions and google docs. These can allow some forms of collaboration with code editing or at least exchanges of iterative coding done locally and remotely. This document written as a quick trial run to get that up and running.

The exported table of contents below presents a tentative pipeline of our workflow, which we also discussed editing and rearranging in places for improved parsimony. For peer-reviewing, we can comment on these Rmarkdown reports directly to the pdf attachment using the usual callout boxes and we can also edit the backend code and push these commits to the github repository for downloading locally or forking remotely This gives us a kind of double-layered privacy so that no html.docs or data sources are available beyond the repo access. When cloning the github repo [@hester] you will find .gitignore rules that include that "Data/" folder otherwise shared via the project drive. Worth noting that for reducing word limit not all backend code was made visible in the report.

1 Import

1.1 Import permanent sample plot data

Import permanent sample plot data in csv format. Here and for other data sourcing we need to set our working directory to where the csv dataset is stored in our local folder. That is because when working with Rmarkdown.Rmd files and github repositories, the working directory will reset back to the root folder where the repo is synced to once the code chunk is passed. This means we can store the data privately twice over, so no raw data is saved on github nor on any shared drive folder. Though, shared static data sources are recommended for click-&-run outputs (need to discuss). This is done using the following code:

```
library(readr)
library(tibble)
faib_psp <- read.csv("./Data/FAIB_PSP_20211028.csv")
print(as_tibble(faib_psp), n = 10)
```

```
## # A tibble: 5,916 x 73
##    row_id clstr_id     samp_id meas_no meas_yr meas_first meas_last   tsa tsa_no
##     <int> <chr>        <chr>     <int>   <int> <chr>      <chr>     <int>  <int>
##  1      1 55023G00050~ 55023 ~       0    1992 Y          N            26     26
##  2      2 55023G00050~ 55023 ~       0    1992 Y          N            26     26
##  3      3 55023G00050~ 55023 ~       0    1992 Y          N            26     26
##  4      4 55023G00050~ 55023 ~       0    1992 Y          N            26     26
##  5      5 55023G00050~ 55023 ~       0    1992 Y          N            26     26
##  6      6 55023G00050~ 55023 ~       1    2002 N          Y            26     26
##  7      7 55023G00050~ 55023 ~       1    2002 N          Y            26     26
##  8      8 55023G00050~ 55023 ~       1    2002 N          Y            26     26
##  9      9 55023G00050~ 55023 ~       1    2002 N          Y            26     26
## 10     10 55023G00050~ 55023 ~       1    2002 N          Y            26     26
## # ... with 5,906 more rows, and 64 more variables: mgmt_unit <chr>,
## #   sampletype <chr>, project_design <chr>, bgc_zone <chr>, ysm_main <chr>,
## #   mat_main <lgl>, species_class <chr>, meas_dt <chr>, no_meas <int>,
## #   period <int>, tot_period <int>, tfl <int>, no_plots <int>, own_sched <chr>,
## #   own_sched_descrip <chr>, samp_sts <chr>, grid_size <lgl>, grid_base <lgl>,
## #   protect_psp <chr>, utm_source <chr>, utm_zone <int>, utm_easting <int>,
## #   utm_northing <int>, bcalb_x <dbl>, bcalb_y <dbl>, aspect <int>, ...
```

```
#DT::datatable(faib_psp, rownames = FALSE, filter="top", options = list(pageLength = 5, scrollX=T))
```

1.2 Import AOI and VRI layers to derive bbox and species raster

For AOI shapefile downloads, you can find the working link for direct download+imports from the BC Geographic Warehouse (BCGW) through the auto-generated email sent from iMapBC with subject line 'assembled'. You need to open up the webpage that the emailed link forwards to and then copy the working link from your browser. In chrome, this takes a third step. Its easy once you look for the link that has the extension '.zip' at the end. Since the GeoBC provide temporary links with short life-span, I've also added code for using cloud links from Cabin account; that is in case client requires a complete click-and-play deliverable.

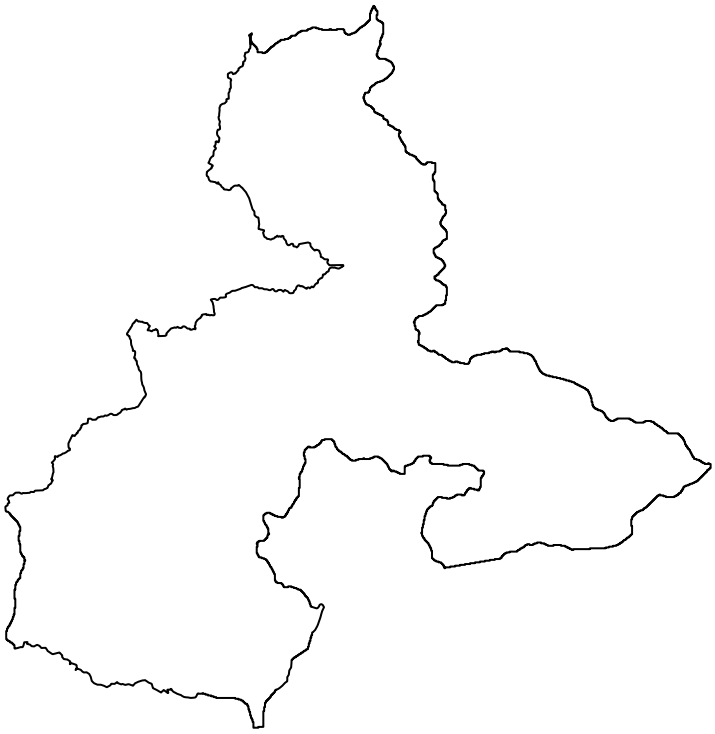
For VRI downloads, instead of the BCGW custom menu, we used the permanent link to the static data source of the Vegetation Resources Inventory dataset here. This includes the usual GIS package of shapefiles and geodatabase which we transform into simple features, from which we then extract our target layers using the 'subset' function. It is useful to consider and treat simple features just as dataframe objects but with more features/information attached like sticky geometry, table rules or matrix indices.

R doesn't seem to like interruptions during zip downloading so best to make sure internet connection is steady or super fast. The VRI zip 'Rank 1" is 4.7GB. Same goes for provincial wildfire dataset, which was subsetted to include post-1999 fire occurances. One last thing to mention, after some trial and error I found that for any markdown rendering or long script reruns, dplyr and any other conflicting packages needs to be loaded last just before their operations and that all dplyr operations need to be contained within the same code chunk as below. Once a new package is laoded afterwards, dplyr becomes problematic and sends inccorrect error messages. A new package 'conflict' helps coerce the machine to the prompted package function that is followed by two colons

* Chapter 4 from Pebezma's manual on 'Simple Features in R' link here

* Chapter 3 from Robin LoveLace's book 'Geocomputation in R' link here

```
library(conflicted)
conflict_prefer("select", "raster")
library(sf)
library(sp)
library(dplyr)
#download.file(url = "https://distribution.data.gov.bc.ca/BCGW_7113060B_1643392193299_1360.zip",
 #              destfile = "./Data/aoi_boundary.zip",
  #             overwrite=TRUE)
#zip_file_aoi = ("./Data/aoi_boundary.zip")
#unzip(zip_file_aoi, exdir="./Data", overwrite = TRUE)
aoi_sf = read_sf("./Data/BCTS_OPERATING_AREAS_SP/BCTS_OP_AR_polygon.shp")
aoi_sf = rename(aoi_sf, AOI_Boundary = SHAPE)
aoi_sf = aoi_sf[1, "AOI_Boundary"]
plot(aoi_sf)
```



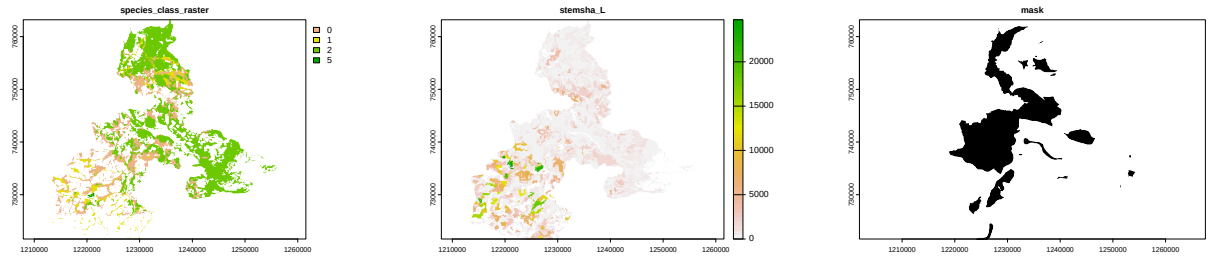

```
#download.file(url = "https://pub.data.gov.bc.ca/datasets/02dba161-fdb7-48ae-a4bb-bd6ef017c36d/2019/VEG
#zip_file_vri = ("./Data/vri_layers.zip")
#unzip(zip_file_vri, exdir="./Data", overwrite = TRUE)
#download.file(url = "https://distribution.data.gov.bc.ca/BCGW_7113060B_1644439767665_15956.zip", destf
#zip_file_wildfire = ("./Data/wildfire_mask.zip")
#unzip(zip_file_wildfire, exdir="./Data", overwrite = TRUE)
wildfire_sf = read_sf("./Data/PROT_HISTORICAL_FIRE_POLYS_SP//H_FIRE_PLY_polygon.shp")
vri_sf = read_sf("./Data/VEG_COMP_LYR_R1_POLY/VEG_R1_PLY_polygon.shp")
vri_species = vri_sf["SPEC_CD_1"]
vri_stemsha = vri_sf["LIVE_STEMS"]
vri_harvest = vri_sf["HRVSTDT"]
wildfire_sf = wildfire_sf["FIRE_YEAR"]
wildfire_aoi = wildfire_sf$FIRE_YEAR > 2000
wildfire_aoi = st_intersection(st_make_valid(wildfire_sf), aoi_sf)
vri_species_aoi = st_intersection(st_make_valid(vri_species), aoi_sf)
vri_species_aoi$SPEC_CD_1 = as.factor(vri_species_aoi$SPEC_CD_1)
vri_species_aoi =  dplyr::filter(vri_species_aoi, SPEC_CD_1 == "PL" | SPEC_CD_1 == "SB" | SPEC_CD_1 == 
  SPEC_CD_1 == "SX" | SPEC_CD_1 == "FD" | SPEC_CD_1 == "CW" | SPEC_CD_1 == "HW" | SPEC_CD_1 == "BL")
vri_species_aoi$species_class = dplyr::recode(vri_species_aoi$SPEC_CD_1,
             PL = 0, PLI = 0, SB = 1, SE = 1, SX = 1, FD = 2, FDI = 2,CW = 3, HW = 4, BL = 5)
summary.factor(vri_species_aoi$species_class)
```

```
##    0    1    2    5
##  979  433 1841    1
```

```
vri_stemsha_aoi = st_intersection(st_make_valid(vri_stemsha), aoi_sf)
vri_stemsha_aoi = rename(vri_stemsha_aoi, stemsha_L = LIVE_STEMS)
stemsha_L_sf = vri_stemsha_aoi["stemsha_L"]
species_class_sf = vri_species_aoi["species_class"]
```

For rasteriize operations, we generated a spatRaster template and fitted using the line 'touches' argument to avoid any void edges. Two good resources on manipulating simple features available just below.

```
library(raster)
library(terra)
library(rgdal)
raster_template = rast(ext(species_class_sf), resolution = 20, crs = st_crs(species_class_sf)$wkt)
species_class_rast = rasterize(vect(species_class_sf),
  raster_template, field = "species_class", touches = TRUE)
stemsha_L_rast = rasterize(vect(stemsha_L_sf),
  raster_template, field = "stemsha_L", fun = sum, touches = TRUE)
mask_rast = rasterize(vect(wildfire_aoi),
  raster_template, field = "FIRE_YEAR", touches = TRUE)

plot(species_class_rast, main = "species_class_raster")
plot(stemsha_L_rast, main = "stemsha_L")
plot(mask_rast, main = "mask", col = "black", legend=FALSE)
```

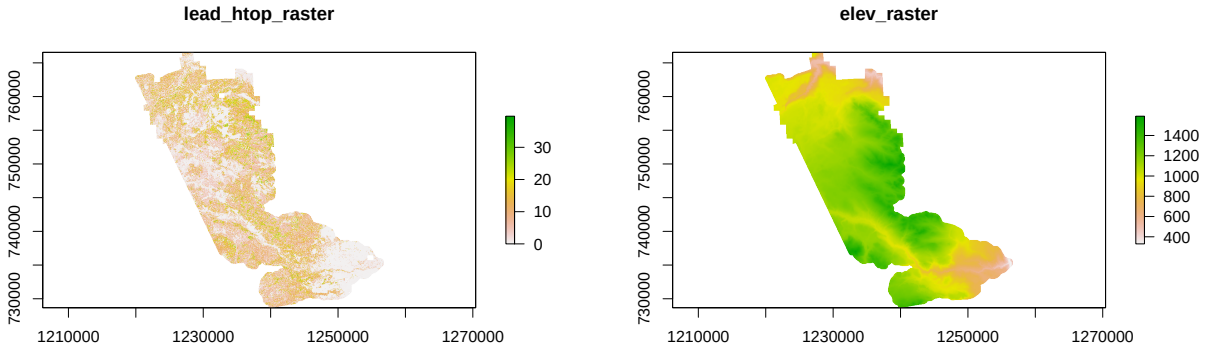


1.3 Import LiDAR data and derive terrain rasters

DEM raster tiles were downloaded from two zipped files. To speed up the markdown output, file.paths were assigned to a zip-directory and unzip-directory. Individual tiles were then assembled as list objects and prepared for merging through two steps. A object was first assigned to the folder location then a second list function was used to index the folder contents as a gridded object. Merging was passed through the *"do.call"* function to generate two raster mosaics, lead_htop and elevation, which were then write as single raster and saved in the Raster_Covariates subfolder. GDAL functions remain in the working script file for further review "0_EFI_CaretEnsemble_ModelTuned_RasterPredicted". See below:

```
# Unpack zipped downloads into assigned directory
#zip_file_vh = ("./Data/VegHt.zip")
#zip_file_be = ("./Data//BareEarth.zip")
#zip_dir_vh = ("./Data/")
#zip_dir_be = ("./Data/")
#unzip(zip_file_vh, exdir=zip_dir_vh, overwrite = TRUE)
#unzip(zip_file_be, exdir=zip_dir_be, overwrite = TRUE)
# Assign sub-directory to where unzipped files become nested
#unzip_dir_vh <- paste0("./Data/VegHt")
```

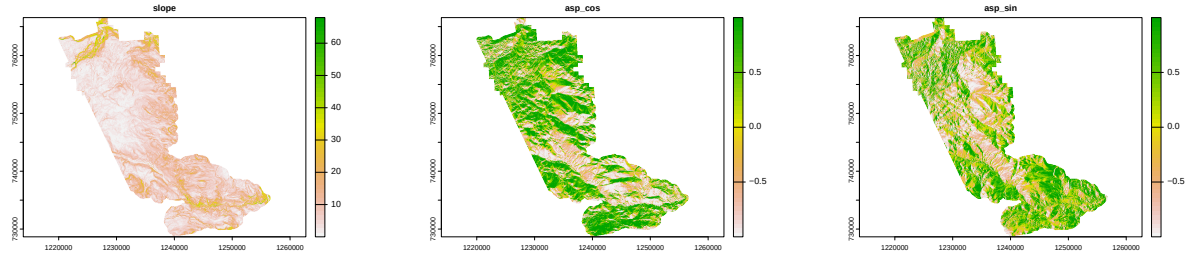
```
#unzip_dir_be <- paste0("./Data/BareEarth")

# Compile folder contents as list objects.
#filez_vh = list.files(
#  unzip_dir_vh,
#  full.names = T,
#  all.files = FALSE,
#  pattern = '.tif$')
#filez_be = list.files(
#  unzip_dir_be,
#  full.names = T,
#  all.files = FALSE,
#  pattern = '.tif$')
# Prepare lists for merging by indexing contents
#lead_htop_raster_list <- lapply(filez_vh, raster)
#elev_raster_list <- lapply(filez_be, raster)

# Merge and assign overlapping tiles tolerance = 1
#lead_htop_raster = do.call(merge, c(lead_htop_raster_list, tolerance = 1))
#elev_raster = do.call(merge, c(elev_raster_list, tolerance = 1))
# Save outputs as new raster.tif files
#writeRaster(lead_htop_raster, filename = "./Data/Raster_Covariates/lead_htop_raster.tif", overwrite=TR
#writeRaster(elev_raster, filename = "./Data/Raster_Covariates/elev_raster.tif", overwrite=TRUE)
lead_htop_raster = raster::raster("./Data/Raster_Covariates/lead_htop_raster.tif")
elev_raster = raster::raster("./Data/Raster_Covariates/elev_raster.tif")
plot(lead_htop_raster, main = "lead_htop_raster")
plot(elev_raster, main = "elev_raster")
```



From the merged elevation raster from above, we derived slope and aspect rasters by applying the nice and easy 'terrain' function from the terra package. The terra package also allowed us to work with spatRasters, which are kind of raster-lite files that function super well and fast in R. After deriving an elevation spatRaster, we kept it as a reference layer over which all other raster (or spatRaster) processing operations were conducted.

One other key strategy to this workflow was that it helped to avoid modifications being made to LiDAR data. Though, some subsequent options still remain uncertain. That is, whether its better to aggregate resolution of LiDAR files from 1m to 20m before or after deriving the landscape metrics. The former is obviously less expensive for running time.

```
elev = rast(elev_raster)
crs(elev) = "epsg:3005"
elev = aggregate(elev, fact = 20, fun = mean)
slope = terrain(elev, v="slope", unit="degrees", neighbors=8)
aspect = terrain(elev, v="aspect", unit="degrees", neighbors=8)
asp_cos = cos((aspect*pi)/180)
asp_sin = sin((aspect*pi)/180)
plot(slope, main = "slope")
plot(asp_cos, main = "asp_cos")
plot(asp_sin, main = "asp_sin")
```



2 Tidy

2.1 Tidy permanent sample plot data

To run the final ecosystem models and generate the predicted rasters, we need to match the number and naming of predictors between our fitted data (permanent sample plot data) and our spatial data (raster stack). From descriptives of category labeling and species lists, the 'spc_live1' predictor in faib dataset was adopted. However, this might need checking. In addition to our target variable 'wsvha_L', the faib data was subsetted down to include only 'elev', 'slope', 'asp_co', 'asp_sin', 'lead_htop', 'stemsha_L' and 'species_class' as predictors. The data was then scanned for missing or problematic observations and transformed into numeric values required for raster operations.

```
faib_psp$spc_live1 = as.factor(faib_psp$spc_live1)
faib_psp = subset(faib_psp, spc_live1 == "PL" | spc_live1 == "SB" | spc_live1 == "SE" |
  spc_live1 == "SX" | spc_live1 == "FD" | spc_live1 == "CW" | spc_live1 == "HW" | spc_live1 == "BL")
faib_psp$species_class = dplyr::recode(faib_psp$spc_live1,
    PL = 0, SB = 1, SE = 1, SX = 1, FD = 2, CW = 3, HW = 4, BL = 5)
faib_psp$asp_cos = cos((faib_psp$aspect * pi) / 180)
faib_psp$asp_sin = sin((faib_psp$aspect * pi) / 180)
faib_psp$wsvha_L = as.numeric(faib_psp$wsvha_L)
faib_psp$stemsha_L = as.numeric(faib_psp$stemsha_L)
faib_psp$slope = as.numeric(faib_psp$slope)
faib_psp$aspect = as.numeric(faib_psp$aspect)
faib_psp$asp_cos = as.numeric(faib_psp$asp_cos)
faib_psp$asp_sin = as.numeric(faib_psp$asp_sin)
faib_psp$lead_htop = as.numeric(faib_psp$lead_htop)
faib_psp$species_class = as.numeric(faib_psp$species_class)
faib_psp$elev = as.numeric(faib_psp$elev)
```

The data was subsetted and cleaned twice over, first for model 1 (incl. stemsha_L) and again for model 2 (excl. stemsha_L) before the model fitting stage. There's likely a tidier way to do this, but havent seen

the light yet. Please rearrange as you see fit. Also, for use in spatial partitioning or mlr packages, the final dataframe was also promoted to SpatialPointsDataFrame and simplefeature towards end of code chunk (l.259)

```
faib_vri_true_m1_df = faib_psp[
  c("elev", "slope", "asp_cos", "asp_sin", "lead_htop", "species_class", "stemsha_L", "wsvha_L")]
faib_vri_true_m2_df = faib_psp[
  c("elev", "slope", "asp_cos", "asp_sin", "lead_htop", "species_class", "wsvha_L")]
faib_vri_true_m1_df$lead_htop[faib_vri_true_m1_df$lead_htop < 1.3] = NA
faib_vri_true_m2_df$lead_htop[faib_vri_true_m2_df$lead_htop < 1.3] = NA
faib_vri_true_m1_df = na.omit(faib_vri_true_m1_df)
faib_vri_true_m2_df = na.omit(faib_vri_true_m2_df)
sum(is.na(faib_vri_true_m1_df))
```

```
## [1] 0
```

```
sum(is.na(faib_vri_true_m2_df))
```

```
## [1] 0
```

```
print(as_tibble(faib_vri_true_m2_df), n = 10)
```

```
## # A tibble: 5,264 x 7
##     elev slope   asp_cos asp_sin lead_htop species_class wsvha_L
##    <dbl> <dbl>     <dbl>   <dbl>     <dbl>         <dbl>   <dbl>
##  1   793    15 -1.84e-16      -1      23.0             0    310.
##  2   793    15 -1.84e-16      -1      23.0             0    309.
##  3   793    15 -1.84e-16      -1      23.0             0    308.
##  4   793    15 -1.84e-16      -1      23.0             0    303.
##  5   793    15 -1.84e-16      -1      23.0             0    288.
##  6   793    15 -1.84e-16      -1      26.0             0    390.
##  7   793    15 -1.84e-16      -1      26.0             0    390.
##  8   793    15 -1.84e-16      -1      26.0             0    389.
##  9   793    15 -1.84e-16      -1      26.0             0    384.
## 10   793    15 -1.84e-16      -1      26.0             0    369.
## # ... with 5,254 more rows
```

```
faib_vri_true_sf = st_as_sf(faib_psp, coords = c("bcalb_x", "bcalb_y"), crs = 3153)
faib_vri_true_sp = as(faib_vri_true_sf, "Spatial")
```

Permanent sample plot data was split using a 80:20 ratio to derive training and test sets for model validation. This was repeated three times both for model1 and model2 so that an X and y array was generate for each, as well as a complete dataframe split. The X and y arrays, which assigned the target variable its own split and predictors another provided quicker run times in subsequent modelling operations.

```
n <- nrow(faib_vri_true_m1_df)
frac <- 0.8
ix <- sample(n, frac * n)
train_m1 = faib_vri_true_m1_df[ix,]
test_m1 = faib_vri_true_m1_df[-ix,]
train_m2 = faib_vri_true_m2_df[ix,]
```

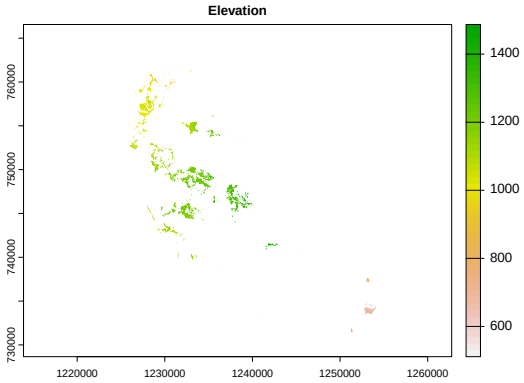
```
test_m2 = faib_vri_true_m2_df[-ix,]

X_train_m1=train_m1[,-8]
X_test_m1=test_m1[,-8]
y_train_m1=train_m1[,8]
y_test_m1=test_m1[,8]

X_train_m2=train_m2[,-7]
X_test_m2=test_m2[,-7]
y_train_m2=train_m2[,7]
y_test_m2=test_m2[,7]

X_m1 = faib_vri_true_m1_df[,-8]
y_m1 = faib_vri_true_m1_df[,8]
X_m2 = faib_vri_true_m2_df[,-7]
y_m2 = faib_vri_true_m2_df[,7]
```

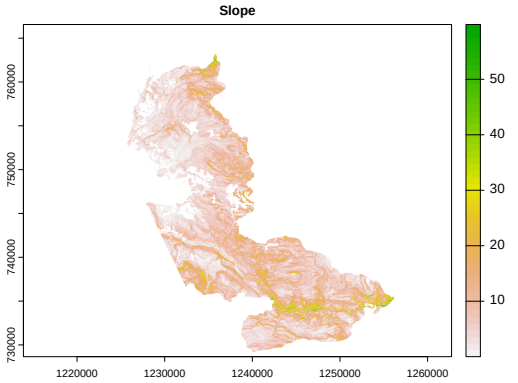
2.2 Tidy raster covariates

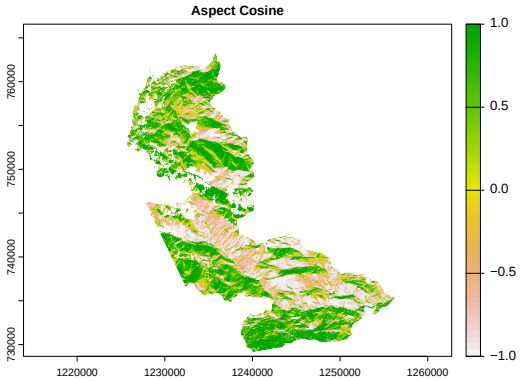
Using terra functions and the elevation spatRaster created above, raster covariates were resampled, clipped, and reprojected to fit the LiDAR data. This was faster than the 'raster' and 'stars' package and produced fewer margin errors and data voids compared to QuantumGIS and Python pipeline we tested. First the LiDAR rasters were reprojected from EPSG:9001 to 3005, and then the 'resample' tool was used to run all processing at once: i.e. matching origin, crs, res and extents.

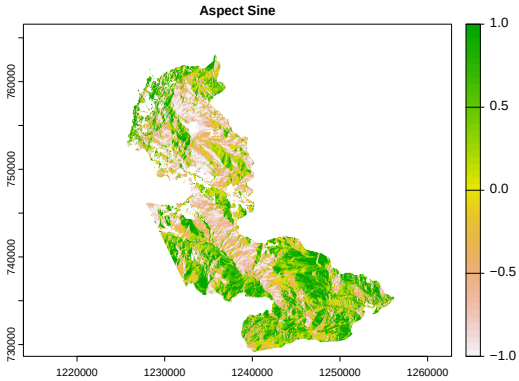
```
lead_htop = rast(lead_htop_raster)
stems = stemsha_L_rast
species = species_class_rast
mask = mask_rast

crs(lead_htop) = "epsg:3005"
lead_htop = aggregate(lead_htop, fact = 20, fun = mean)
slope = terra::resample(slope, elev, method="bilinear")
asp_cos = terra::resample(asp_cos, elev, method="bilinear")
asp_sin = terra::resample(asp_sin, elev, method="bilinear")
lead_htop = terra::resample(lead_htop, elev, method="bilinear")
species = terra::resample(species, elev, method="near")
stems = terra::resample(stems, elev, method="bilinear")
mask = terra::resample(mask, elev, method="near")

elev = mask(elev, vect(aoi_sf))
lead_htop = mask(lead_htop, vect(aoi_sf))
slope = mask(slope, elev, inverse=FALSE)
asp_cos = mask(asp_cos, elev, inverse=FALSE)
asp_sin = mask(asp_sin, elev, inverse=FALSE)
species = mask(species, elev, inverse=FALSE)
stems = mask(stems, elev, inverse=FALSE)

lead_htop[lead_htop < 1.3] <- NA
mask_lead_htop = mask(elev, lead_htop, inverse=TRUE)
mask = mask(mask_lead_htop, mask, inverse=FALSE)
elev = mask(elev, mask, inverse=FALSE)
```

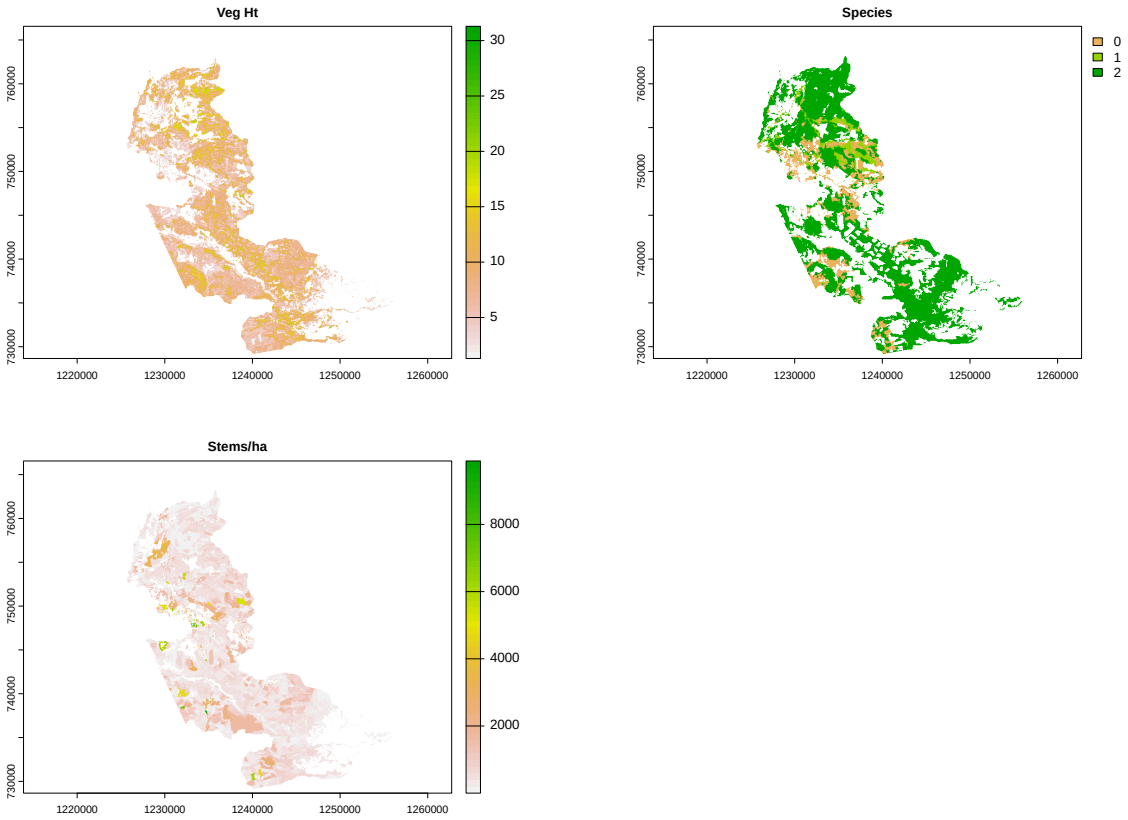
```
lead_htop = mask(lead_htop, mask, inverse=TRUE)
slope = mask(slope, mask, inverse=TRUE)
asp_cos = mask(asp_cos, mask, inverse=TRUE)
asp_sin = mask(asp_sin, mask, inverse=TRUE)
species = mask(species, mask, inverse=TRUE)
stems = mask(stems, mask, inverse=TRUE)

plot(elev, main="Elevation")
plot(slope, main="Slope")
plot(asp_cos, main="Aspect Cosine")
plot(asp_sin, main="Aspect Sine")
plot(lead_htop, main="Veg Ht")
plot(species, main="Species")
plot(stems, main="Stems/ha")
```











Spatial covariates were then transformed back from spatRasters to rasters and assembled as raster stacks (covs_m1 and covs_m2). Maybe worth checking again here the naming of rasters before converting and stacking. If needed, the following chunk can be used to rename spatRasters:

```
names(elev) = "elev"
names(slope) = "slope"
names(asp_cos) = "asp_cos"
names(asp_sin) = "asp_sin"
names(species) = "species_class"
names(stems) = "stemsha_L"
names(lead_htop) = "lead_htop"

elev_raster = raster::raster(elev)
slope_raster = raster::raster(slope)
asp_cos_raster = raster::raster(asp_cos)
asp_sin_raster = raster::raster(asp_sin)
species_class_raster = raster::raster(species)
stemsha_L_raster = raster::raster(stems)
lead_htop_raster = raster::raster(lead_htop)

#writeRaster(slope_raster, filename = "./Data/Raster_Covariates/slope_raster.tif", overwrite=TRUE)
#writeRaster(asp_cos_raster, filename = "./Data/Raster_Covariates/asp_cos_raster.tif", overwrite=TRUE)
#writeRaster(asp_sin_raster, filename = "./Data/Raster_Covariates/asp_sin_raster.tif", overwrite=TRUE)
#writeRaster(species_class_raster, filename = "./Data/Raster_Covariates/species_class_raster.tif", over
#writeRaster(stemsha_L_raster, filename = "./Data/Raster_Covariates/stemsha_L_raster.tif", overwrite=TR
```

```
covs_m1 = stack(elev_raster, slope_raster, asp_cos_raster, asp_sin_raster,
                lead_htop_raster, species_class_raster,stemsha_L_raster)
covs_m2 = stack(elev_raster,slope_raster, asp_cos_raster, asp_sin_raster,
                lead_htop_raster, species_class_raster)

names(covs_m2)
```

```
## [1] "elev"          "slope"         "asp_cos"       "asp_sin"
## [5] "lead_htop"     "species_class"
```

```
names(covs_m1)
```

```
## [1] "elev"          "slope"         "asp_cos"       "asp_sin"
## [5] "lead_htop"     "species_class" "stemsha_L"
```

```
names(faib_vri_true_m2_df)
```

```
## [1] "elev"          "slope"         "asp_cos"       "asp_sin"
## [5] "lead_htop"     "species_class" "wsvha_L"
```

```
names(faib_vri_true_m1_df)
```

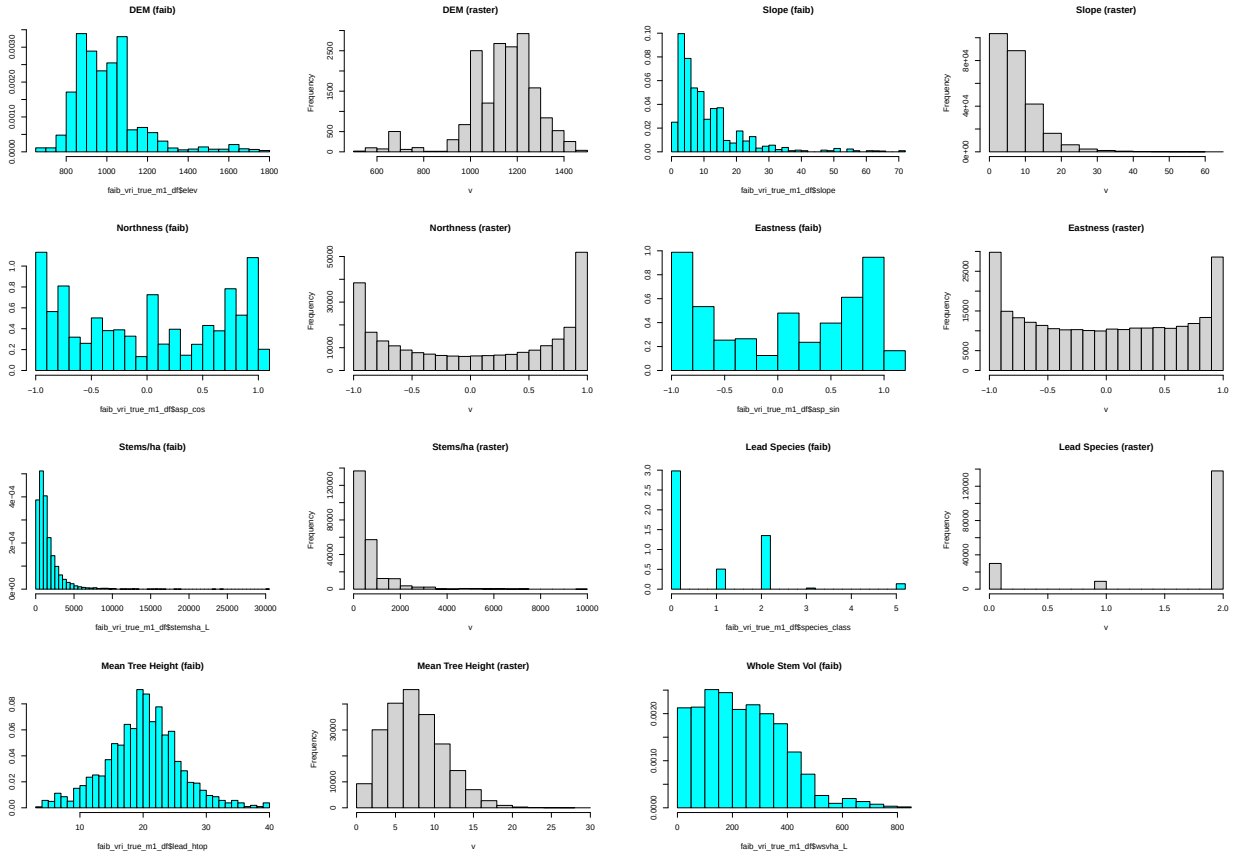
```
## [1] "elev"          "slope"         "asp_cos"       "asp_sin"
## [5] "lead_htop"     "species_class" "stemsha_L"     "wsvha_L"
```

3 Transform

Data was explored first by comparing visually the distribution of raster covariates and faib dataframe predictors. Wilcoxon normality test of signed-rank sum was reported accordingly. Visualization of data distribution was compared using simple histograms from the 'MASS' package and base R functions. Pixel inclusion was increased to widen raster sample (22000000). When working with larger Williams Lake-wide rasters, exploratory data analysis may require an improved sampling strategy to account for larger pixel count.

```
library(MASS)
wilcox.test(faib_vri_true_m1_df$elev) # p<0.0001
wilcox.test(faib_vri_true_m1_df$slope) # p<0.0001
wilcox.test(faib_vri_true_m1_df$asp_cos) # p=0.8749
wilcox.test(faib_vri_true_m1_df$asp_sin) # p<0.0001
wilcox.test(faib_vri_true_m1_df$lead_htop) # p<0.0001
wilcox.test(faib_vri_true_m1_df$stemsha_L) # p<0.0001
wilcox.test(faib_vri_true_m1_df$wsvha_L) # p<0.0001

truehist(faib_vri_true_m1_df$elev, main="DEM (faib)", maxpixels=22000000)
hist(elev, main="DEM (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$slope, main="Slope (faib)", maxpixels=22000000)
hist(slope, main="Slope (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$asp_cos, main="Northness (faib)", maxpixels=22000000)
hist(asp_cos, main="Northness (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$asp_sin, main="Eastness (faib)", maxpixels=22000000)
```

```
hist(asp_sin, main="Eastness (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$stemsha_L, main="Stems/ha (faib)", maxpixels=22000000)
hist(stems, main="Stems/ha (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$species_class, main="Lead Species (faib)", maxpixels=22000000)
hist(species, main="Lead Species (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$lead_htop, main="Mean Tree Height (faib)", maxpixels=22000000)
hist(lead_htop, main="Mean Tree Height (raster)", maxpixels=22000000)
truehist(faib_vri_true_m1_df$wsvha_L, main="Whole Stem Vol (faib)", maxpixels=22000000)
```



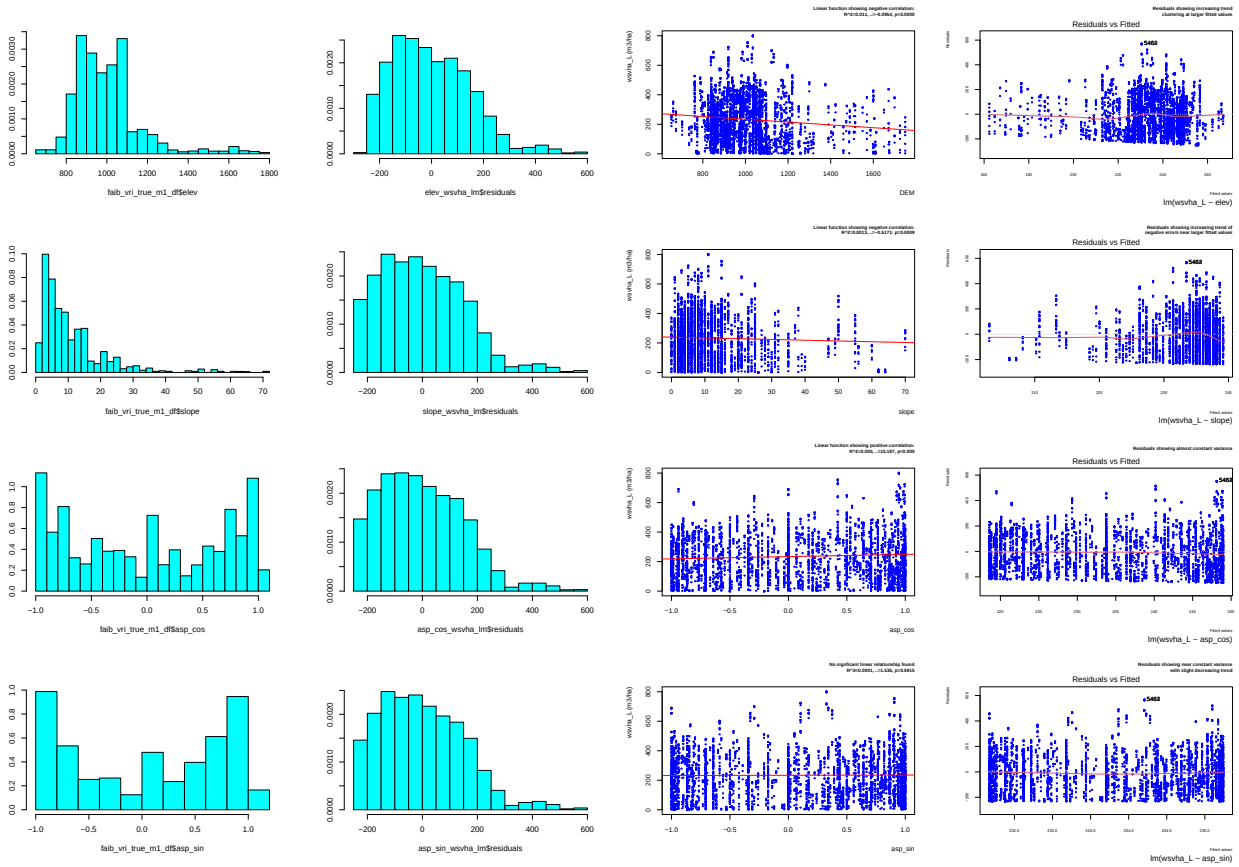
3.1 Explore data transformations

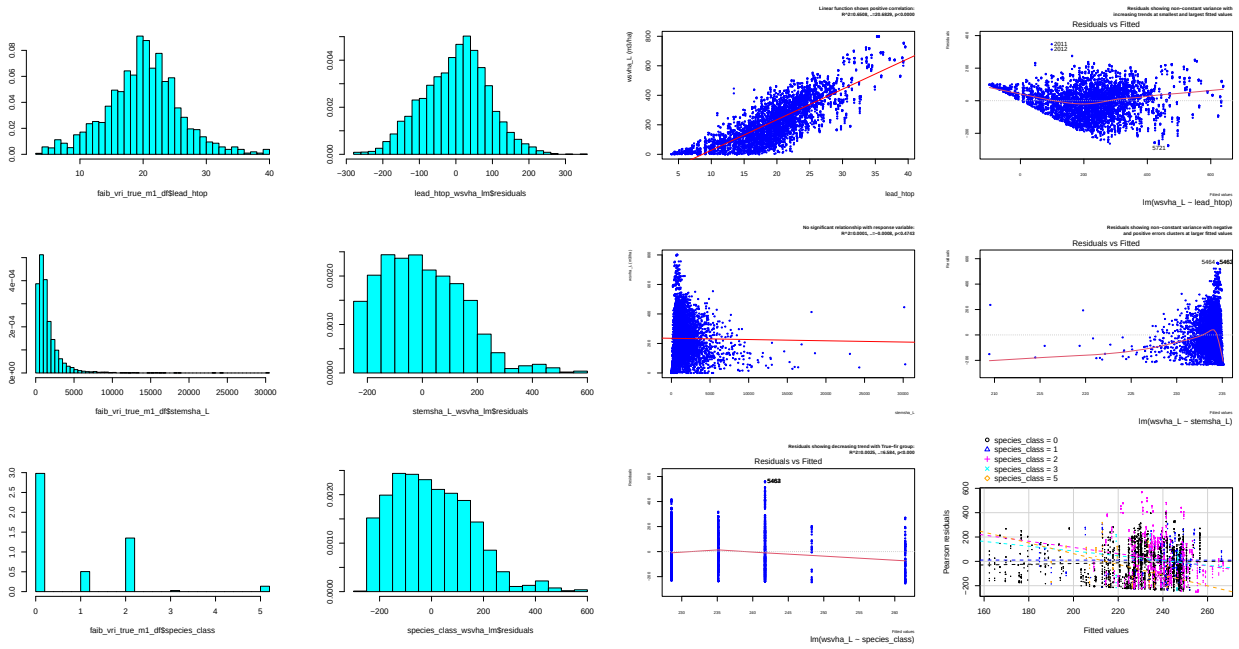
Faib predictors were also tested for linear hypotheses and emerging trends in residual variance. To examine linearity and assess predictor influence, predictor variables were fitted with univariate linear functions. Using these predictive functions, residuals were mapped and Breush-Pagan test of constant variance were reported.

```
library(olsrr)
library(car)
elev_wsvha_lm = lm(wsvha_L ~ elev, data = faib_vri_true_m1_df)
slope_wsvha_lm = lm(wsvha_L ~ slope, data = faib_vri_true_m1_df)
asp_cos_wsvha_lm = lm(wsvha_L ~ asp_cos, data = faib_vri_true_m1_df)
asp_sin_wsvha_lm = lm(wsvha_L ~ asp_sin, data = faib_vri_true_m1_df)
lead_htop_wsvha_lm = lm(wsvha_L ~ lead_htop, data = faib_vri_true_m1_df)
species_class_wsvha_lm = lm(wsvha_L ~ species_class, data = faib_vri_true_m1_df)
stemsha_L_wsvha_lm = lm(wsvha_L ~ stemsha_L, data = faib_vri_true_m1_df)
```

```
ols_test_breusch_pagan(elev_wsvha_lm) # p=0.0012
ols_test_breusch_pagan(slope_wsvha_lm) # p=0.6685
ols_test_breusch_pagan(asp_cos_wsvha_lm) # p<0.000000
ols_test_breusch_pagan(asp_sin_wsvha_lm) # p=0.5316
ols_test_breusch_pagan(lead_htop_wsvha_lm) # p<0.000000000000000
ols_test_breusch_pagan(stemsha_L_wsvha_lm) # p<0.000000000000000

truehist(faib_vri_true_m1_df$elev)
truehist(elev_wsvha_lm$residuals)
plot(wsvha_L ~ elev, data = faib_vri_true_m1_df,
 main="Linear function showing negative correlation:\nR^2=0.011,  =-0.0954, p<0.0000",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.8, cex.axis=0.8, adj=1,
     ylab = "wsvha_L (m3/ha)", xlab = "DEM")
abline(elev_wsvha_lm, col = "red")
plot(elev_wsvha_lm, which=1,
 main="Residuals showing increasing trend\n clustering at larger fitted values",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
truehist(faib_vri_true_m1_df$slope)
truehist(slope_wsvha_lm$residuals)
plot(wsvha_L ~ slope, data = faib_vri_true_m1_df,
 main="Linear function showing negative correlation:\nR^2=0.0013,  =-0.5171: p=0.0009",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.8, cex.axis=0.8, adj=1,
     ylab = "wsvha_L (m3/ha)", xlab = "slope")
abline(slope_wsvha_lm, col = "red")
plot(slope_wsvha_lm, which=1,
 main="Residuals showing increasing trend of\nnegative errors near larger fitted values",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
truehist(faib_vri_true_m1_df$asp_cos)
truehist(asp_cos_wsvha_lm$residuals)
plot(wsvha_L ~ asp_cos, data = faib_vri_true_m1_df,
 main="Linear function showing positive correlation:\nR^2=0.005,  =15.197, p<0.000",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.8, cex.axis=0.8, adj=1,
     ylab = "wsvha_L (m3/ha)", xlab = "asp_cos")
abline(asp_cos_wsvha_lm, col = "red")
plot(asp_cos_wsvha_lm, which=1,
 main="Residuals showing almost constant variance",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
truehist(faib_vri_true_m1_df$asp_sin)
truehist(asp_sin_wsvha_lm$residuals)
plot(wsvha_L ~ asp_sin, data = faib_vri_true_m1_df,
 main="No signficant linear relationship found\n R^2<0.0001,  =1.535, p=0.5915",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.8, cex.axis=0.8, adj=1,
     ylab = "wsvha_L (m3/ha)", xlab = "asp_sin")
abline(asp_sin_wsvha_lm, col = "red")
plot(asp_sin_wsvha_lm, which=1,
 main="Residuals showing near constant variance\n with slight decreasing trend",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
truehist(faib_vri_true_m1_df$lead_htop)
truehist(lead_htop_wsvha_lm$residuals)
plot(wsvha_L ~ lead_htop, data = faib_vri_true_m1_df,
 main="Linear function shows positive correlation:\n R^2=0.6508,  =20.6829, p<0.0000",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.8, cex.axis=0.8, adj=1,
     ylab = "wsvha_L (m3/ha)", xlab = "lead_htop")
```

```
abline(lead_htop_wsvha_lm, col = "red")
plot(lead_htop_wsvha_lm, which=1,
 main="Residuals showing non-constant variance with\n increasing trends at smallest and largest fitted v
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
truehist(faib_vri_true_m1_df$stemsha_L)
truehist(stemsha_L_wsvha_lm$residuals)
plot(wsvha_L ~ stemsha_L, data = faib_vri_true_m1_df,
 main="No significant relationship with response variable:\nR^2=0.0001,  =-0.0008, p<0.4743",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1,
     ylab = "wsvha_L (m3/ha)", xlab = "stemsha_L")
abline(stemsha_L_wsvha_lm, col = "red")
plot(stemsha_L_wsvha_lm, which=1,
 main="Residuals showing non-constant variance with negative\nand positive errors clusters at larger fi
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
truehist(faib_vri_true_m1_df$species_class)
truehist(species_class_wsvha_lm$residuals)
plot(species_class_wsvha_lm, which=1,
 main="Residuals showing decreasing trend with True-fir group:\nR^2=0.0025,  =6.584, p<0.000",
  col="blue", pch=20, cex=0.5, cex.main=0.6, cex.lab=0.5, cex.axis=0.5, adj=1)
car::residualPlots(elev_wsvha_lm, terms= ~ 1 | species_class, cex=0.1, pch=19)
```





3.2 Apply data transformations

Six predictors exhibited non-normal distributions with left-leaning skewness (W, p<0.001) and five predictors produced non-constant variance against the response variable (B, p<0.001). Residuals showed increasing trends and patterns of funnelling that suggested anomalies were clustered among larger fitted values or among stands with higher wsvha values. In addition, significantly negative influences were observed by slope and stemsha on wsvha, while initial modelling generated negative value estimates confirming need for data normalizations (wsvha<0.00m3/ha). In response, modelled data was treated with three Preprocess functions from the caret package. The 'center' method was used to subtract the mean of the predictors data from the predictor values and the 'scale' method was used to divide them by their standard deviation. A 'BoxCox' transformation applied an exponential lambda to positive values to coerce a Gaussian distribution. Data transformations were implementated iteratively with each model fitting using 'caret' model tuning functions as shown in the following section.

4 Model

Model 1 and model 2 were fitted with the ransformed 'pre-processed' permanent sample plot data and calibrated with nine different algorithms (Table 1). Models were trained using a 10 k-fold cross validation technique, which divided the dataset into 10 groups of 10 data blocks to generate aggregated estimates from across the 100 data folds.

Performance metrics were reported regarding Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Root Mean Squared Ratio (RMSEratio) in order to account for overall model accuracy, level of model precision, and model bias, respectively. Using these metrics, algorithms were optimized using hyper-parameter tuning. An inital deployment of models An pilot deployment of 10k-fold model training and validation was first carried without any repeat training (Table 1). A second deployment was conducted to compare these results with 10k-fold-5repeats in which training folds were initiated from different fold origins during each repeat (Table 2).

Two Support Vector Machine (SVM) algorithms were fitted with a radial and linear kernel, which were tuned using the same tuning grid that tested for optimal cost values in the range of between 1 and 20

	Full Model					Cross Validation		
Model1	Hyperparameter Tuning			MAE	RMSE	MAE	RMSE	$RMSE^{ratio}$
M1.svm.radial$^{\varepsilon}$	ε = 0.02	C = 20	γ = 0.5	3.524	5.877	5.751	9.901	0.594
M1.svm.radial	ε = 0.10	C = 20	γ = 0.5	9.424	10.830	10.801	13.961	0.776
M1.svm.linear	ε = 0.10	C = 01	γ = 1.0	30.000	41.173	170.590	213.531	0.193
M1.glm.caret	ε = 1e-08	C = 25		30.715	40.902	174.112	216.489	0.189
M1.RF.e1071	Mtry = 3	Ntree = 50		3.662	5.576	8.460	12.841	0.420
M1.RF.caret	Mtry = 3	Ntree = 500		3.388	5.178	171.073	215.336	0.024
M1.Epanech	OV = 4.35	Bws = aic		0.226	0.434	176.299	219.720	0.002
M1.LocalConst	OV = 4.35	Bws = leastsq		0.226	0.454	89.042	159.294	0.003
M1.ensemble	α = 0.10	$\hat{\lambda}$ = 0.11031		8.876	13.702			
Model2								
M2.svm.radial$^{\varepsilon}$	ε = 0.02	C = 05	γ = 0.5	2.570	3.541	6.321	14.505	0.244
M1.svm.radial	ε = 0.10	C = 20	γ = 0.5	9.146	10.559	11.694	18.327	0.576
M2.svm.linear	ε = 0.10	C = 01	γ = 1.0	27.910	38.471	174.306	216.880	0.177
M2.glm.caret	ε = 1e-08	C = 25		28.591	37.921	174.485	216.916	0.175
M2.RF.e1071	Mtry = 3	Ntree = 50		3.857	5.709	8.928	13.587	0.420
M2.RF.caret	Mtry = 3	Ntree = 500		3.537	5.271	175.490	218.750	0.024
M2.ensemble	α = 0.10	$\hat{\lambda}$ = 0.11025		9.423	14.463			

Figure 1: Table 1: Hyperparameter tuning and model performance metrics; MAE: Mean absolute error, RMSE: Root squared mean error, RMSEratio: Root squared mean error ratio (RMSEfull/RMSEcv), Mtry: Number of variables at each split, Ntree: maximum number of decisions trees = Epsilon, = Gamma, C = Cost; = Alpha, = Lambda.

and scanned for optimal gamma values between -1 and +1. Random Forest (RF) models were calibrated with two hyperparameters using a grid search of Mtry between 2 and 10 variables at each split over two regression trees consisting of 50 and 500 decision branches. An EnsembleElastic-Net model was fitted with a generalized additive linearized model, which was tuned based on optomized results of three foundational models including linear model,

4.1 Model 1: 'M1.svm.radial. '

```
library(e1071)
library(caret)
library(caretEnsemble)
library(randomForest)
library(DescTools)
library(ModelMetrics)
# full-fitted
tuneResult_svm_m2_full_eps <- tune(svm, X_m2, y_m2,
  ranges = list(epsilon = seq(0.02,0.1,0.2), cost = c(1,5,7,15,20), gamma = 2^(-1:1)),
  tunecontrol = tune.control(cross = 10),
  preProcess = c("BoxCox","center","scale"))
tunedModel_svm_m2_full_eps <- tuneResult_svm_m2_full_eps$best.model
save(tunedModel_svm_m2_full_eps, file = "./Results/tunedModel_svm_m2_full_eps.RData")
tunedModel_svm_m2_eps = predict(
  tunedModel_svm_m2_full_eps,
  X_m2, y_m2)
# train-fitted
tuneResult_svm_m2_train_eps <- tune(svm, X_train_m2, y_train_m2,
  ranges = list(epsilon = seq(0.02,0.1,0.2), cost = c(5,7,15,20), gamma = 2^(-1:1)),
```

```
  tunecontrol = tune.control(cross = 10),
  preProcess = c("BoxCox","center","scale"))
tunedModel_svm_m2_train_eps <- tuneResult_svm_m2_train_eps$best.model
# test-fitted
tunedModel_svm_m2_test_eps = predict(tuneResult_svm_m2_train_eps,X_test_m2, y_test_m2)
#performance metrics
tunedModel_svm_m2_test_eps_MAE = MAE(tunedModel_svm_m2_test_eps, y_test_m2)
tunedModel_svm_m2_test_eps_RMSE = RMSE(tunedModel_svm_m2_test_eps, y_test_m2)
tunedModel_svm_m2_full_eps_MAE = MAE(tunedModel_svm_m2_eps, y_m2)
tunedModel_svm_m2_full_eps_RMSE = RMSE(tunedModel_svm_m2_eps, y_m2)
tunedModel_svm_m2_full_epsRMSEratio = tunedModel_svm_m2_full_eps_RMSE/tunedModel_svm_m2_test_eps_RMSE
print(summary(tunedModel_svm_m2_full))
```

4.2 Model 1: 'M1.svm.radial'

```
# full-fitted
tuneResult_svm_m2_full <- tune(svm, X_m2, y_m2,
  ranges = list(cost = c(1,5,7,15,20), gamma = 2^(-1:1)),
  tunecontrol = tune.control(cross = 10,  nrepeat = 5),
  preProcess = c("BoxCox","center","scale"))
tunedModel_svm_m2_full <- tuneResult_svm_m2_full$best.model
save(tunedModel_svm_m2_full, file = "./Models/tunedModel_svm_m2_full.RData")
tunedModel_svm_m2 = predict(tunedModel_svm_m2_full, X_m2, y_m2)
# train-fitted
tuneResult_svm_m2_train <- tune(
  svm, X_train_m2, y_train_m2,
  ranges = list(cost = c(1,5,7,15,20), gamma = 2^(-1:1)),
  tunecontrol = tune.control(cross = 10,  nrepeat = 5),
  preProcess = c("BoxCox","center","scale"))
tunedModel_svm_m2_train <- tuneResult_svm_m2_train$best.model
# test-fitted
tunedModel_svm_m2_test = predict(tunedModel_svm_m2_train, X_test_m2, y_test_m2)
#performance metrics
tunedModel_svm_m2_test_MAE = MAE(tunedModel_svm_m2_test, y_test_m2)
tunedModel_svm_m2_test_RMSE = RMSE(tunedModel_svm_m2_test, y_test_m2)
tunedModel_svm_m2_full_MAE = MAE(tunedModel_svm_m2, y_m2)
tunedModel_svm_m2_full_RMSE = RMSE(tunedModel_svm_m2, y_m2)
tunedModel_svm_m2_full_RMSEratio = tunedModel_svm_m2_full_RMSE/tunedModel_svm_m2_test_RMSE
print(summary(tunedModel_svm_m2_full))
```

4.3 Model 1: 'M1.svm.linear'

```
# full fitted
tuneResult_svmLinear_m2_10k_full <- train(wsvha_L~.,data=faib_vri_true_m2_df,
  trControl = model_training_10fold_parallel,
  method = 'svmLinear', metric = 'RMSE', tuneLength = 10,
  preProcess = c("BoxCox",'center', 'scale'))
tunedModel_svmLinear_m2 = predict(tuneResult_svmLinear_m2_10k_full, data = faib_vri_true_m2_df)
# train fitted
```

```
tuneResult_svmLinear_m2_10k_train <- train(X_train_m2, y_train_m2,
  trControl = model_training_10fold_parallel,
  method = 'svmLinear', metric = 'RMSE', tuneLength = 10,
  preProcess = c("BoxCox",'center', 'scale'))
# test fitted
tunedModel_svmLinear_m2_test = predict(tuneResult_svmLinear_m2_10k_train, data = test_m2)
# performance metrics
tunedModel_svmLinear_m2_test_MAE = mae(tunedModel_svmLinear_m2_test, test_m2$wsvha_L)
tunedModel_svmLinear_m2_test_RMSE = rmse(tunedModel_svmLinear_m2_test, test_m2$wsvha_L)
tunedModel_svmLinear_m2_MAE = mae(tunedModel_svmLinear_m2, faib_vri_true_m2_df$wsvha_L)
tunedModel_svmLinear_m2_RMSE = rmse(tunedModel_svmLinear_m2, faib_vri_true_m2_df$wsvha_L)
tunedModel_svmLinear_m2_RMSEratio = tunedModel_svmLinear_m2_RMSE/tunedModel_svmLinear_m2_test_RMSE
```

4.4 Model 1: 'M1.glm.caret'

```
#full-fitted
tuneResult_GLM_m2_full <- train(wsvha_L~., data=faib_vri_true_m2_df,
  trControl = model_training_10fold_parallel,
  method = 'glm', metric = 'RMSE', tuneLength = 10,
  preProcess = c("BoxCox",'center', 'scale'))
tunedModel_GLM_m2_full <- tuneResult_GLM_m2_full$finalModel
tunedModel_GLM_m2 = predict(tunedModel_GLM_m2_full, data=faib_vri_true_m2_df, type = "response")
#train-fitted
tuneResult_GLM_m2_train <- train(wsvha_L ~., data=train_m2,
  trControl = model_training_10fold_parallel,
  method = 'glm', metric = 'RMSE', tuneLength = 10,
  preProcess = c("BoxCox",'center', 'scale'))
tunedModel_GLM_m2_train <- tuneResult_GLM_m2_train$finalModel
#test-fitted
tunedModel_GLM_m2_test = predict(tunedModel_GLM_m2_train, data = test_m2, type = "response")
# performance metrics
tunedModel_GLM_m2_test_MAE = mae(tunedModel_GLM_m2_test, test_m2$wsvha_L)
tunedModel_GLM_m2_test_RMSE = rmse(tunedModel_GLM_m2_test, test_m2$wsvha_L)
tunedModel_GLM_m2_full_MAE = mae(tunedModel_GLM_m2, faib_vri_true_m2_df$wsvha_L)
tunedModel_GLM_m2_full_RMSE = rmse(tunedModel_GLM_m2, faib_vri_true_m2_df$wsvha_L)
tunedModel_GLM_m2_full_RMSEratio = tunedModel_GLM_m2_full_RMSE/tunedModel_GLM_m2_test_RMSE
print(summary(tunedModel_GLM_m2_full))
```

4.5 Model 1: 'M1.RF.e1071'

```
# full-fitted
tuneResult_rf_m2_full <- tune.randomForest(X_m2, y_m2,
  mtry = c(2:10), ntree = 50,
  tunecontrol = tune.control(sampling = "cross", cross = 10),
  preProcess = c("BoxCox","center","scale"))
tunedModel_rf_m2_full <- tuneResult_rf_m2_full$best.model
save(tunedModel_rf_m2_full, file = "./Results/tunedModel_rf_m2_full.RData")
tunedModel_rf_m2 = predict(tunedModel_rf_m2_full,X_m2, y_m2,type = "response")
# train-fitted
```

```
tuneResult_rf_m2_train <- tune.randomForest(X_train_m2, y_train_m2,
  mtry = c(2:10), ntree = 50,
  tunecontrol = tune.control(sampling = "cross", cross = 10),
  preProcess = c("BoxCox","center","scale"))
tunedModel_rf_m2_train <- tuneResult_rf_m2_train$best.model
print(summary(tunedModel_rf_m2_train))
# test-fitted
tunedModel_rf_m2_test = predict(tunedModel_rf_m2_train, X_test_m2, y_test_m2, type="response")
# performance metrics
tunedModel_rf_m2_test_MAE = MAE(tunedModel_rf_m2_test, y_test_m2)
tunedModel_rf_m2_test_RMSE = RMSE(tunedModel_rf_m2_test, y_test_m2)
tunedModel_rf_m2_full_MAE = MAE(tunedModel_rf_m2, y_m2)
tunedModel_rf_m2_full_RMSE = RMSE(tunedModel_rf_m2, y_m2)
tunedModel_rf_m2_full_RMSEratio = tunedModel_rf_m2_full_RMSE/tunedModel_rf_m2_test_RMSE
print(summary(tunedModel_rf_m2_full))
```

4.6 Model 1: 'M1.RF.caret'

```
# full fitted
tuneResult_rfCaret_m2_10k_full <- train(wsvha_L~.,
  data=faib_vri_true_m2_df,
  trControl = model_training_10fold_parallel,
  method = 'rf', metric = 'RMSE', tuneLength = 10,
  preProcess = c("BoxCox",'center', 'scale'))
save(tunedModel_rfCaret_m2_10k, file = "./Results/tunedModel_rfCaret_m2_10k.RData")
tunedModel_rfCaret_m2_10k = tuneResult_rfCaret_m2_10k_full$finalModel
tunedModel_rfCaret_m2 = predict(tuneResult_rfCaret_m2_10k_full, data = faib_vri_true_m2_df)
# train fitted
tuneResult_rfCaret_m2_10k_train = train(
  X_train_m2, y_train_m2,
  trControl = model_training_10fold_parallel,
  method = 'rf',
  metric = 'RMSE',
  tuneLength = 10,
  preProcess = c("BoxCox",'center', 'scale'))
#test fitted
tunedModel_rfCaret_m2_test = predict(tuneResult_rfCaret_m2_10k_train, data = test_m2)
# performance metrics
tunedModel_rfCaret_m2_test_MAE = mae(tunedModel_rfCaret_m2_test, test_m2$wsvha_L)
tunedModel_rfCaret_m2_test_RMSE = rmse(tunedModel_rfCaret_m2_test, test_m2$wsvha_L)
tunedModel_rfCaret_m2_MAE = mae(tunedModel_rfCaret_m2, faib_vri_true_m2_df$wsvha_L)
tunedModel_rfCaret_m2_RMSE = rmse(tunedModel_rfCaret_m2, faib_vri_true_m2_df$wsvha_L)
tunedModel_rfCaret_m2_RMSEratio = tunedModel_rfCaret_m2_RMSE/tunedModel_rfCaret_m2_test_RMSE
print(summary(tunedModel_rfCaret_m2))
```

4.7 Model 1: 'M1.ensemble'

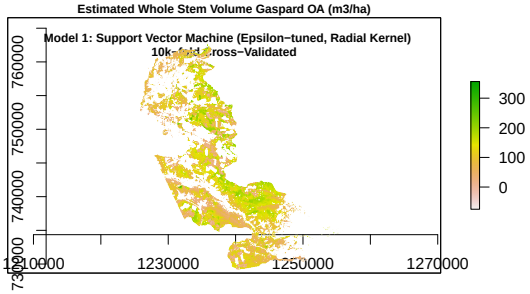
```
# model stack fitted
model_list_m2_10k <- caretList(X_train_m2, y_train_m2,
```

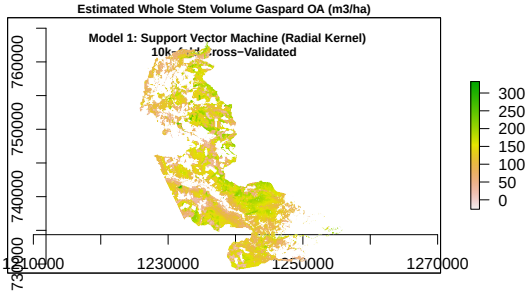
```
  trControl = model_training_10fold_parallel,
  methodList = c('glm', 'svmLinear', 'rf'),
  tuneLength=10, continue_on_fail = FALSE,
  preProcess = c('BoxCox', 'center','scale'))
model_results_m2_10k <- data.frame(
  GLM = min(model_list_m2_10k$glm$results$RMSE),
  SVMLINEAR = min(model_list_m2_10k$svmLinear$results$RMSE),
  RF = min(model_list_m2_10k$rf$results$RMSE))
resamples_m2_10k = resamples(model_list_m2_10k)
# model ensemble fitted
stack_m2_10k = caretStack(model_list_m2_10k,
  method = 'glmnet', metric = 'RMSE',
  trControl = trainControl(method = "repeatedcv",
  savePredictions = "final", summaryFunction=defaultSummary), tuneLength=6)
# performance metrics
plot(stack_m2_10k)
print(stack_m2_10k)
```

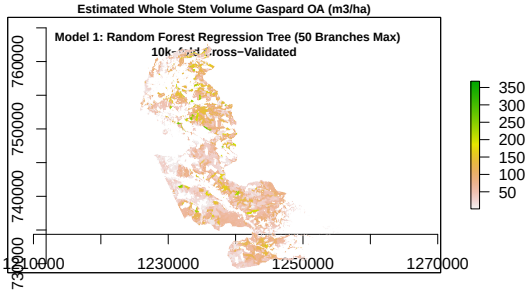
5 Visualize

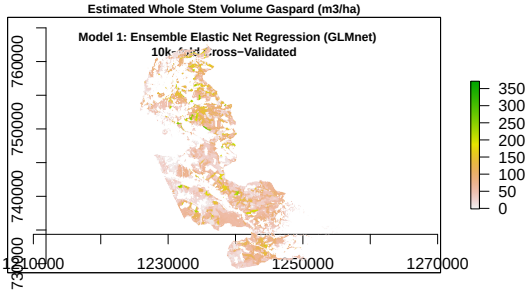
Four best-performing models, including the Random Forest 50-Tree model, the two Support Vector Radial Kernel models, and the Ensemble Elastic Net Model, were applied to covariate stacks and used to spatially predict a wsvha raster. Outputs were saved in GeoTiff format in the 'Results' local folder available on project drive and were presented below for visual comparison.

```
library(prettymapr)
par(mfrow = c(2, 2))
tunedModel_svm_m2_eps_to_raster <- predict(covs_m2, tunedModel_svm_m2_full_eps)
writeRaster(tunedModel_svm_m2_eps_to_raster, filename = "./Results/Rasters/tunedModel_svm_m2_eps_to_ras
tunedModel_svm_m2_eps_to_raster_plot = plot(tunedModel_svm_m2_eps_to_raster,
  main= "Estimated Whole Stem Volume Gaspard OA (m3/ha)\n
  Model 1: Support Vector Machine (Epsilon-tuned, Radial Kernel)\n10k-fold Cross-Validated", cex.main =

tunedModel_svm_m2_to_raster <- predict(covs_m2, tunedModel_svm_m2_full)
writeRaster(tunedModel_svm_m2_to_raster, filename = "./Results/Rasters/tunedModel_svm_m2_to_raster.tif"
tunedModel_svm_m2_to_raster_plot = plot(tunedModel_svm_m2_eps_to_raster,
  main= "Estimated Whole Stem Volume Gaspard OA (m3/ha)\n
  Model 1: Support Vector Machine (Radial Kernel)\n10k-fold Cross-Validated", cex.main = 0.75, line = -

tunedModel_rf_m2_to_raster <- predict(covs_m2, tunedModel_rf_m2_full)
writeRaster(tunedModel_rf_m2_to_raster, filename = "./Results/Rasters/tunedModel_rf_m2_to_raster.tif",
tunedModel_rf_m2_to_raster_plot = plot(tunedModel_rf_m2_to_raster,
  main= "Estimated Whole Stem Volume Gaspard OA (m3/ha)\n
  Model 1: Random Forest Regression Tree (50 Branches Max)\n10k-fold Cross-Validated", cex.main = 0.75,

caretensemble_m2_10k_to_raster <- predict(covs_m2, stack_m2_10k)
writeRaster(caretensemble_m2_10k_to_raster, filename = "./Results/Rasters/caretensemble_m2_10k_to_raste
caretensemble_m2_10k_to_raster_plot = plot(caretensemble_m2_10k_to_raster,
  main="Estimated Whole Stem Volume Gaspard (m3/ha)\n
  Model 1: Ensemble Elastic Net Regression (GLMnet)\n10k-fold Cross-Validated", cex.main = 0.75, line=
```


Estimated Whole Stem Volume Gaspard OA (m3/ha)
Model 1: Support Vector Machine (Epsilon−tuned, Radial Kernel)
10k−fold Cross−Validated
760000
750000
740000
730000
1210000
1230000
1250000
1270000
300
200
100
0


Estimated Whole Stem Volume Gaspard OA (m3/ha)
Model 1: Support Vector Machine (Radial Kernel)
10k−fold Cross−Validated
760000
750000
740000
730000
1210000
1230000
1250000
1270000
300
250
200
150
100
50
0


Estimated Whole Stem Volume Gaspard OA (m3/ha)
Model 1: Random Forest Regression Tree (50 Branches Max)
10k−fold Cross−Validated
760000
750000
740000
730000
1210000
1230000
1250000
1270000
350
300
250
200
150
100
50


Estimated Whole Stem Volume Gaspard (m3/ha)
Model 1: Ensemble Elastic Net Regression (GLMnet)
10k−fold Cross−Validated
760000
750000
740000
730000
1210000
1230000
1250000
1270000
350
300
250
200
150
100
50
0