Forest Data Analysis Report

Jakob Danel Federick Bruch

2024-01-23

Contents

1 Introduction

This report documents the analysis of forest data for different tree species.

2 Methods

2.1 Data acquisition

Our primary objective is to identify patches where one tree species exhibits a high level of dominance, striving to capture monocultural stands within the diverse forests of Nordrhein-Westfalia (NRW). Recognizing the practical challenges of finding true monocultures, we aim to identify patches where one species is highly dominant, enabling meaningful comparisons across different species.

The study is framed within the NRW region due to the availability of an easily accessible dataset. Our focus includes four prominent tree species in NRW: oak, beech, spruce, and pine, representing the most prevalent species in the region. To ensure the validity of our findings, we derive three patches for each species, thereby confirming that observed variables are characteristic of a particular species rather than a specific patch. Each patch is carefully selected to encompass an area of approximately 50-100 hectares and contain between 5,000 and 10,000 trees. Striking a balance between relevance and manageability, these patches avoid excessive size to enhance the likelihood of capturing varied species mixes and ensure compatibility with local hardware.

Specific Goals:

1. Retrieve patches with highly dominant tree species.
2. Minimize or eliminate the presence of human-made structures within the selected patches.

To achieve our goals, we utilized the waldmonitor dataset (Welle et al. 2022) and the map provided by (Blickensdoerfer 2022), both indicating dominant tree species in NRW. We identified patches of feasible size where both sources predicted the presence of a specific species. Further validation involved examining sentinel images of these forest regions to assess the evenness of structures, leaf color distribution, and the absence of significant human-made structures such as roads or buildings. The subsequent preprocessing steps, detailed in the following subsection, involved refining our selected patches and deriving relevant variables, such as tree distribution and density, to ensure that the chosen areas align with the desired research domains.

2.2 Preprocessing

In this research study, the management and processing of a large dataset are crucial considerations. The dataset's substantial size necessitates careful maintenance to ensure efficient handling. Furthermore, the data should be easily processable and editable to facilitate necessary corrections and precalculations within the context of our research objectives. To achieve our goals, we have implemented a framework that automatically derives data based on a shapefile, delineating areas of interest. The processed data and results of precalculations are stored

in a straightforward manner to enhance accessibility. Additionally, we have designed functions that establish a user-friendly interface, enabling the execution of algorithms on subsets of the data, such as distinct species. These interfaces are not only directly callable by users but can also be integrated into other functions to automate processes. The overarching aim is to streamline the entire preprocessing workflow using a single script, leveraging only the shapefile as a basis. This subsection details the accomplishments of our R-package in realizing these goals, outlining the preprocessing steps undertaken and justifying their necessity in the context of our research.

The data are stored in a data subdirectory of the root directory in the format `species/location-name/tile-nam`
To automate the matching of areas of interest with the catalog from the Land NRW[1], we utilize the intersecting tool developed by Heisig[2]. This tool, allows for the automatic retrieval and placement of data downloaded from the Land NRW catalog. To enhance data accessibility, we have devised an object that incorporates species, location name, and tile name (the NRW internal identifier) for each area This object facilitates the specification of the area to be processed. Additionally, we have defined an initialization function that downloads all tiles, returning a list of tile location objects for subsequent processing. A pivotal component of the package's preprocessing functionality is the map function, which iterates over a list of tile locations (effectively the entire dataset) and accepts a processing function as an argument. The subsequent paragraph outlines the specific preprocessing steps employed, all of which are implemented within the mapping function.

To facilitate memory-handling capabilities, each of the tiles, where one area can span multiple tiles, has been split into manageable chunks. We employed a 50x50m size for each tile, resulting in the division of original 1km x 1km files into 400 tiles. These tiles are stored in our directory structure, with each tile housed in a directory named after its tile name and assigned an id as the filename. Implementation-wise, the `lidr::catalog_retile` function was instrumental in achieving this segmentation. The resulting smaller chunks allow for efficient iteration during subsequent preprocessing steps.

The next phase involves reducing our data to the actual size by intersecting the tiles with the defined area of interest. Using the `lidR::merge_spatial` function, we intersect the area derived from the shapefile, removing all point cloud items outside this region. Due to our tile-wise approach, empty tiles may arise, and in such cases, those tiles are simply deleted.

Following the size reduction to our dataset, the next step involves correcting the `z` values. The `z` values in the data are originally relative to the ellipsoid used for referencing, but we require them to be relative to the ground. To achieve this, we utilize the `lidR::tin` function, which extrapolates a convex hull between all ground points (classified by the data provider) and calculates the z value based on this structure.

Subsequently, we aim to perform segmentation for each distinct tree, marking each item of the point cloud with a tree ID. We employ the algorithm described by Li et al. (2012), using pa-

[1]https://www.opengeodata.nrw.de/produkte/geobasis/hm/3dm_l_las/3dm_l_las/, last visited 7th Dec 2023
[2]https://github.com/joheisig/GEDIcalibratoR, last visited 7th Dec 2023

rameters `li2012(dt1 = 2, dt2 = 3, R = 2, Zu = 10, hmin = 5, speed_up = 12)`. The meanings of these parameters are elucidated in Li et al.'s work (Li et al. 2012).

Finally, the last preprocessing step involves individual tree detection, seeking a single `POINT` object for each tree. The `lidR::lmf` function, an implementation of the tree data using a local maximum approach, is utilized for this purpose (Popescu and Wynne 2004). The results are stored in GeoPackage files within our data structure.

See Section 5.1 for the implementation of the preprocessing.

2.3 Analysis of different distributions

Analysis of data distributions is a critical aspect of our research, with a focus on comparing two or more distributions. Our objective extends beyond evaluating the disparities between species; we also aim to assess differences within a species. To gain a comprehensive understanding of the data, we employ various visualization techniques, including histograms, density functions, and box plots.

In tandem with visualizations, descriptive statistics, such as means, standard errors, and quantiles, are leveraged to provide key insights into the central tendency and variability of the data.

For a more quantitative analysis of distribution dissimilarity, statistical tests are employed. The Kullback-Leibler (KL) difference serves as a measure to compare the similarity of a set of distributions. This involves converting distributions into their density functions, with the standard error serving as the bandwidth. The KL difference is calculated for each pair of distributions, as it is asymmetric. For the two distributions the KL difference is defined as following (Kullback 1951):

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

To obtain a symmetric score, the Jensen-Shannon Divergence (JSD) is utilized (Grosse et al. 2002), expressed by the formula:

$$JS(P||Q) = \frac{1}{2} * KL(P||M) + \frac{1}{2} * KL(Q||M)$$

Here, $M = \frac{1}{2} * (P + Q)$. The JSD provides a balanced measure of dissimilarity between distributions (Brownlee 2019). For comparing the different scores to each other, we will use averages.

Additionally, the Kolmogorov-Smirnov Test is implemented to assess whether two distributions significantly differ from each other. This statistical test offers a formal evaluation of the dissimilarity between empirical distribution functions.

3 Results

3.1 Researched areas

```
library(ggplot2)
sf::sf_use_s2(FALSE)
patches <- sf::read_sf("research_areas.shp") |> sf::st_centroid()

de <- sf::read_sf("results/results/states_de/Bundesländer_2017_mit_Einwohnerzahl.shp") # S
nrw <- de[5,] |> sf::st_geometry()


ggplot() + geom_sf(data = nrw) +
    geom_sf(data = patches, mapping = aes(col = species))
```

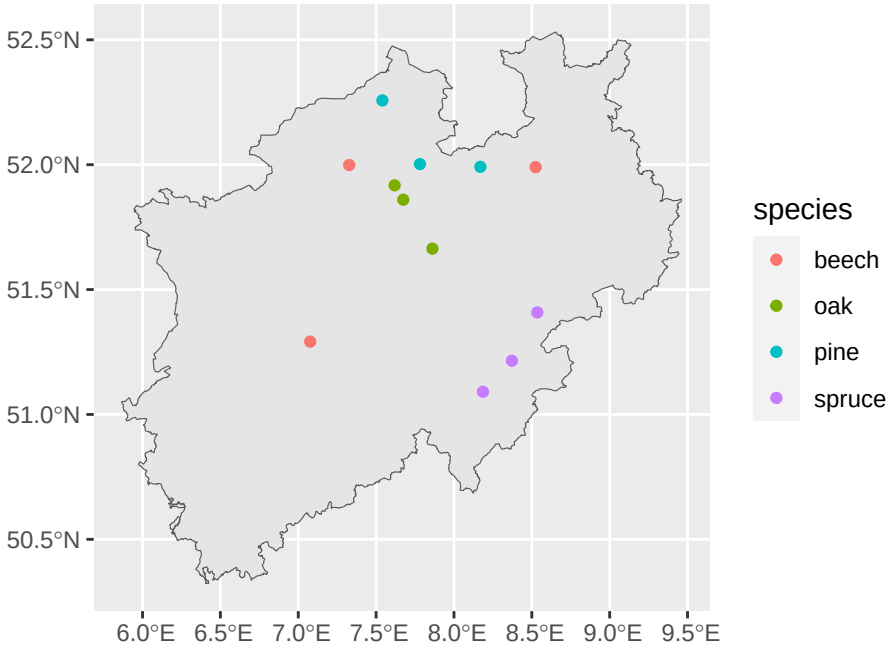

Figure 1: Locations of the different patches with the dominant species for that patch. The patches centroids are displayed on a basemap describing the borders from NRW.

We draw three patches for each species from different regions (see Table 1). We download the LiDAR data for those patches and runned all preprocessing steps as described. We than checked with certain derived parameters (e.g. tree heights, tree distributions or tree density) that all patches contain valid forest data. In that step we discovered, that in one patch some

forest clearance took place in the near past. This patch was removed from the dataset and was replaced with a new one.

In our research, drawing patches evenly distributed across Nordrhein-Westfalia is inherently constrained by natural factors. Consequently, the patches for oak and pine predominantly originate from the Münsterland region, as illustrated in Figure 1. For spruce, the patches were derived from Sauerland, reflecting the prevalence of spruce forests in this specific region within NRW, as corroborated by Welle et al. (Welle et al. 2022) and Blickensdörfer et al. (Blickensdoerfer 2022). Beech patches, on the other hand, were generated from diverse locations within NRW. Across all patches, no human-made objects were identified, with the exception of small paths for pedestrians and forestry vehicles.

The distribution of area and detections is notable for each four species. Beech covers 69,791.9 hectares with a total of 5,954 detections, oak spans 63,232.49 hectares with 5,354 detections, pine extends across 72,862.4 hectares with 8,912 detections, and spruce encompasses 57,940.02 hectares with 8,619 detections. Both the amount of detections and the corresponding area exhibit a relatively uniform distribution across the diverse patches, as summarized in Table 1.

With the selected dataset described, we intentionally chose three patches for each four species that exhibit a practical and usable size for our research objectives. These carefully chosen patches align with the conditions essential for our study, providing comprehensive and representative data for in-depth analysis and meaningful insights into the characteristics of each tree species within the specified areas.

```
shp <- sf::read_sf("research_areas.shp")
table <- lfa::lfa_get_all_areas()

sf::sf_use_s2(FALSE)
for (row in 1:nrow(table)) {
  area <-
    dplyr::filter(shp, shp$species == table[row, "specie"] &
                    shp$name == table[row, "area"])
  area_size <- area |> sf::st_area()
  point <- area |> sf::st_centroid() |> sf::st_coordinates()
  table[row,"point"] <- paste0("(",round(point[1], digits = 4),", ",round(point[2],digits

  table[row, "area_size"] = round(area_size,digits = 2) #paste0(round(area_size,digits = 2

  amount_det <- nrow(lfa::lfa_get_detection_area(table[row, "specie"], table[row, "area"])
  if(is.null(amount_det)){
    cat(nrow(lfa::lfa_get_detection_area(table[row, "specie"], table[row, "area"])),table[
  }
  table[row, "amount_detections"] = amount_det
```

```
  # table[row, "specie"] <- lfa::lfa_capitalize_first_char(table[row,"specie"])
  table[row, "area"] <- lfa::lfa_capitalize_first_char(table[row,"area"])
  }
table$area <- gsub("_", " ", table$area)
table$area <- gsub("ue", "ü", table$area)
table = table[,!names(table) %in% c("specie")]

knitr::kable(table, "html", col.names = c("Patch Name","Location","Area size (m²)","Amount
  kableExtra::kable_styling(
    bootstrap_options = c("striped", "hold_position", "bordered","responsive"),
    stripe_index = c(1:3,7:9),
    full_width = FALSE
  ) |>
  kableExtra::pack_rows("Beech", 1, 3) |>
  kableExtra::pack_rows("Oak", 4, 6) |>
  kableExtra::pack_rows("Pine", 7, 9) |>
  kableExtra::pack_rows("Spruce", 10, 12) |>
  kableExtra::column_spec(1, bold = TRUE)
```

Table 1: Summary of researched patches grouped by species, with their location, area and the amount of detected trees.

Patch Name	Location	Area size (m²)	Amount tree detections
Beech			
Bielefeld brackwede	(8.5244, 51.9902)	161410.57	1443
Billerbeck	(7.3273, 51.9987)	185887.25	1732
Wülfenrath	(7.0769, 51.2917)	350621.21	2779
Oak			
Hamm	(7.8618, 51.6639)	269397.22	2441
Münster	(7.6187, 51.9174)	164116.61	1270
Rinkerode	(7.6744, 51.8598)	198811.09	1643
Pine			
Greffen	(8.1697, 51.9913)	49418.81	513
Mesum	(7.5403, 52.2573)	405072.85	5031
Telgte	(7.7816, 52.0024)	274132.34	3368
Spruce			
Brilon	(8.5352, 51.4084)	211478.20	3342
Oberhundem	(8.1861, 51.0909)	151895.53	2471
Osterwald	(8.3721, 51.2151)	216026.43	2806

3.2 Distribution of the tree heights

```
detections <- lfa::lfa_get_detections()
```

In this study, we scrutinize the distribution of tree heights, focusing initially on the density distribution to unravel the nuances across various tree species. Notably, our examination reveals distinctive patterns, with Oak and Pine exhibiting significantly steeper peaks in their density curves compared to Beech and Spruce. While all species present unique density curves, a commonality emerges—each curve is characterized by a single peak, except for the intriguing exception observed in Telgte. Taking Beech as an illustrative example, our findings indicate a notable shift in the peak to a considerably higher extent. The varinace in the density curves indicating that an differencation between species only with the help oof tree height values could be difficult.

```
lfa::lfa_create_density_plots(detections, value_column = "Z", category_column1 = "area", c
```

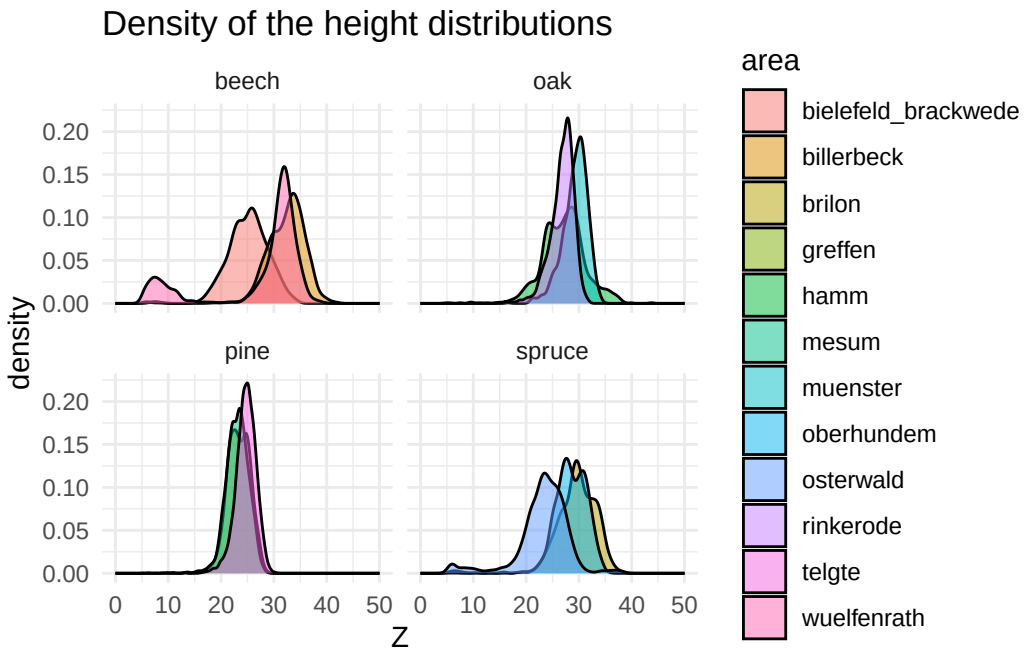

Figure 2: Density of the height distribitions of the detectected trees. Splitted by the different researched areas and grouped by the dominant specie in this area.

To have a deeper look into the distributions of those Z-values we will now also have a look into the boxplots of the height distrubutions in the different areas. Noteworthy observations include the presence of outliers beyond the extended range of the Whisker Antennas (1.5∗IQR)

in all datasets. Of particular interest is the Rinkerode dataset, which exhibits a higher prevalence of outliers in the upper domain. Anomalies in this dataset are attributed to potential inaccuracies, urging a critical examination of data integrity. A pairwise examination of Oak and Pine species indicates higher mean heights for Oak compared to Pine. This insight underscores the significance of species-specific attributes in shaping overall height distributions. Further exploration into the factors contributing to these mean differences enhances our understanding of the unique characteristics inherent to each species. Contrary to expectations, the spread within a particular species does not exhibit significant divergence from the spread observed between different species. This finding suggests that while species-specific traits play a crucial role in shaping height distributions, certain overarching factors may contribute to shared patterns across diverse tree populations.

```
lfa::lfa_create_boxplot(detections, value_column = "Z", category_column1 = "area", categor
```

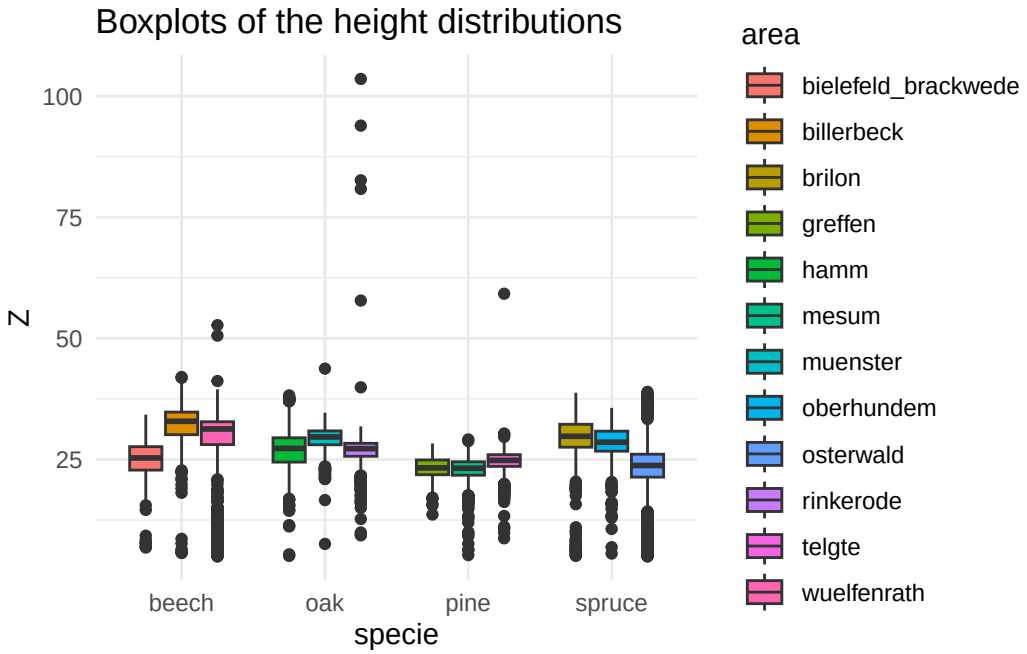

Figure 3: Boxplots of the height distribitions of the detectected trees. Splitted by the different researched areas and grouped by the dominant specie in this area.

Our examination of Kullback-Leibler Divergence (KLD) and Jensen-Shannon Divergence (JSD) metrics reveals low mean values (KLD: 5.252696, JSD: 2.246663) across different species, indicating overall similarity in tree species height distributions. However, within specific species, particularly Pine, higher divergence values (see Table 5 and Table 10) suggest significant intraspecific differences.

Notably, the Spruce species consistently demonstrates low divergence values across all tested areas, implying a high level of explainability. This finding highlights tree height as a reliable indicator for detecting Spruce trees, indicating its potential for accurate species identification in diverse forest ecosystems.

3.3 Distribution of number of returns per detected tree.

```
data <- sf::st_read("data/tree_properties.gpkg")
neighbors <- lfa::lfa_get_neighbor_paths() |> lfa::lfa_combine_sf_obj(lfa::lfa_get_all_are
data = sf::st_join(data,neighbors, join = sf::st_within)
```

Examining the distribution of LiDAR returns per tree is the focus of our current investigation. Initial analysis involves the study of density graphs representing the distribution of LiDAR returns. The density curves for each species exhibit distinct peaks corresponding to their respective species, providing a clear differentiation in LiDAR return patterns. Notably, there is an exception observed in the Brilon patch (Spruce), where the curve deviates, possibly indicative of variations in forest age. A noteworthy trend is the divergent shape of density curves between coniferous and deciduous trees. Conifers exhibit steeper curves, indicating lower density for higher return values compared to deciduous trees. This disparity underscores the potential of LiDAR data to distinguish between tree types based on return density characteristics. In the case of Beech trees, the peaks' heights vary among different curves, suggesting nuanced variations within the species. Despite these differences, all species consistently peak in similar regions, emphasizing the overarching similarities in LiDAR return patterns across diverse tree species.

```
lfa::lfa_create_density_plots(data, value_column = "number_of_returns", category_column1 =
```

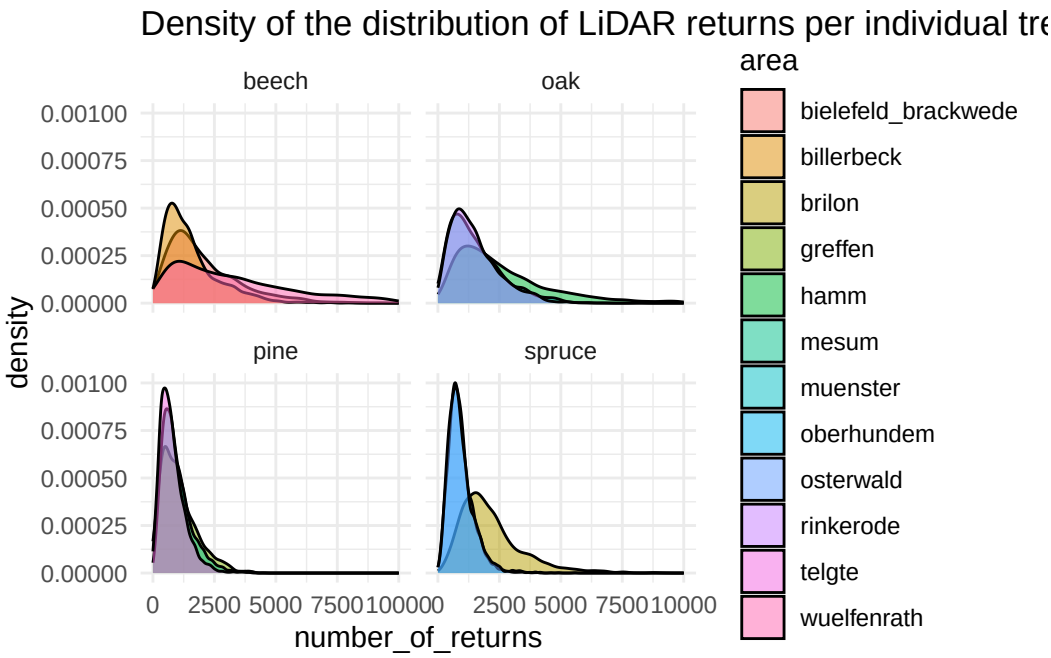

Figure 4: Density of the amount of LiDAR returns per detectected tree. Splitted by the different researched areas and grouped by the dominant specie in this area.

Currently, our investigation focuses on boxplots representing each patch. We observe significant size variations among plots within the same species. Notably, numerous outliers are present above the box in each patch. For Pines, the boxes exhibit a notable similarity. However, the box for Brilon is entirely shifted from other boxes associated with patches featuring Spruce forest.

```
lfa::lfa_create_boxplot(data, value_column = "number_of_returns", category_column1 = "area
```

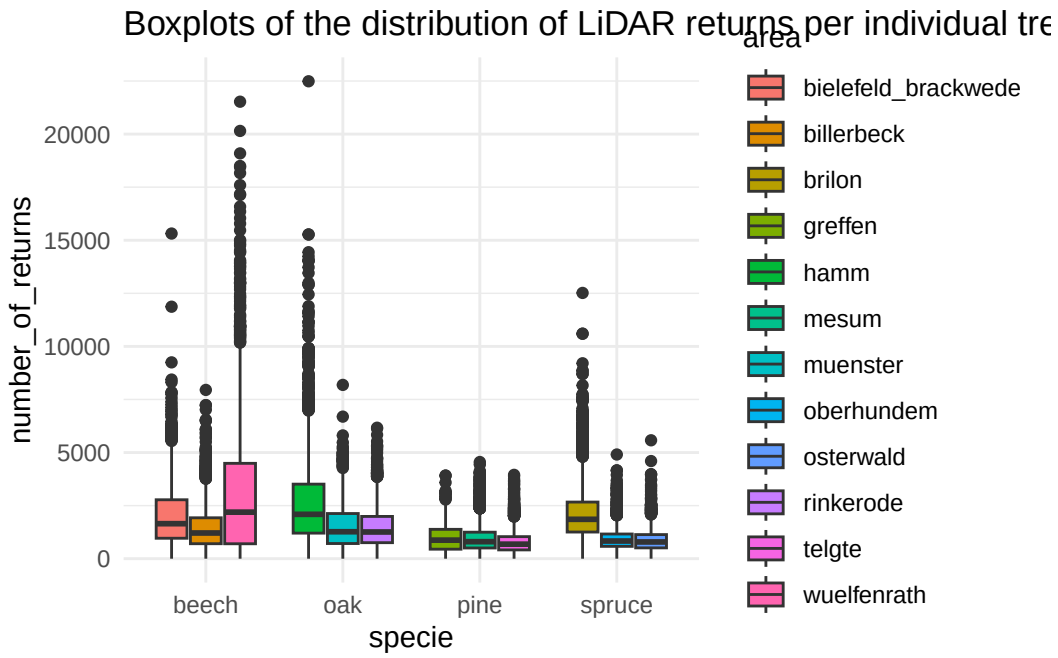

Figure 5: Boxplots of the the amount of LiDAR returns per detectected tree. Splitted by the different researched areas and grouped by the dominant specie in this area.

Overall, our analysis reveals very low results for both Kullback-Leibler Divergence (KLD) and Jensen-Shannon Divergence (JSD) metrics across different species. Within species, there is high explainability observed for the different LiDAR return curves between patches.

This suggests that the number of returns alone may not be a robust predictor for identifying the dominant species in a forest. However, the curves indicate a clear potential for distinguishing between conifers (Pine and Spruce) and deciduous trees (Beech and Oak) based on the number of returns. This observation is further supported by the JSD scores, as detailed in Table 47.

3.4 Density of forest patches

Examining densities provides valuable insights into identifying the dominant species within patches. Spruce stands out as the densest species, surpassing all other patches. Following closely in density is Pine, as depicted in Figure 1 (Figure 6).

Beech and Oak exhibit similar density levels, with Beech consistently denser across all patches. When comparing the highest density patches for each species, Beech consistently outpaces Oak. While Oak is slightly less dense overall ($8.354499 \times 10^{-3} \frac{1}{m^2}$) than Beech ($8.727781 \times 10^{-3} \frac{1}{m^2}$), the distinction in density remains noticeable.

```
library(units)
lfa::lfa_calculate_patch_density() |>
  lfa::lfa_create_grouped_bar_plot(grouping_var = "species", value_col = "density", label_
```

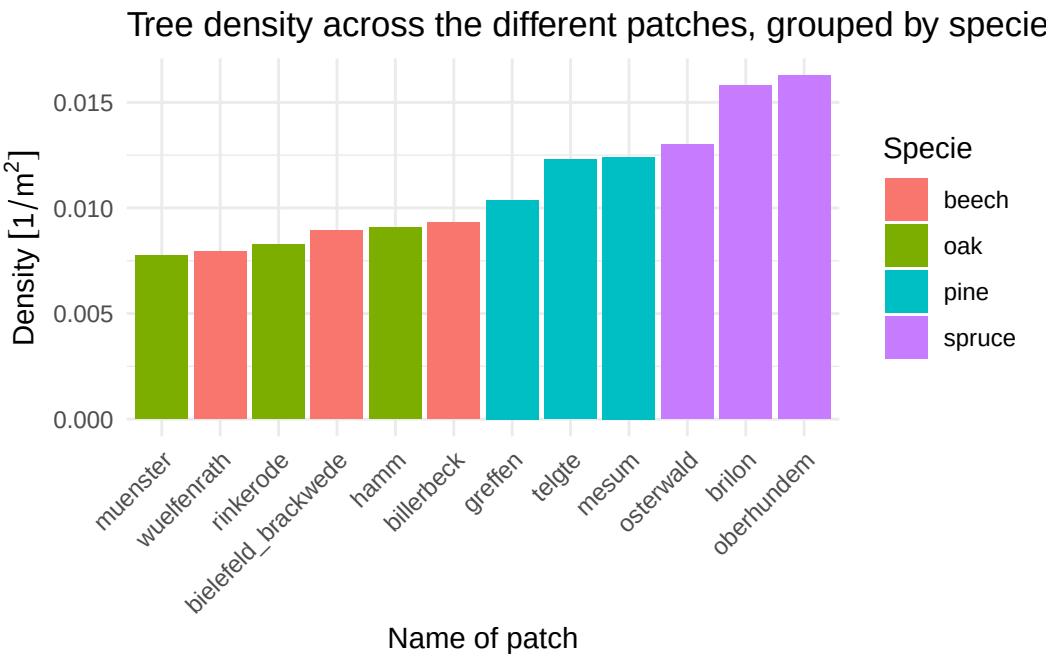

Figure 6: Barplot of the densitys of all patches (#detected trees/area of patch). Colorized by the dominant tree species of each patch.

In summary, our findings indicate that the density of each patch proves highly effective in distinguishing dominant species. Furthermore, the differentiation between conifers (Pine and Spruce) and deciduous trees (Beech and Oak) based on density aligns with patterns observed in the number of return points per detected tree. While distinguishing within conifers is straightforward, discerning between the deciduous tree species Beech and Oak, is possible but poses a moderate challenge.

4 References

Blickensdoerfer, Lukas. 2022. "Dominant Tree Species for Germany (2017/2018)." *Waldatlas-Wald Und Waldnutzung*. Thünen Atlas. https://atlas.thuenen.de/layers/geonode:Dominant_Species_Class.

Brownlee, Jason. 2019. "How to Calculate the KL Divergence for Machine Learning." *MachineLearningMastery.com*. https://machinelearningmastery.com/divergence-between-probability-distributions/.

Grosse, Ivo, Pedro Bernaola-Galván, Pedro Carpena, Ramón Román-Roldán, Jose Oliver, and H Eugene Stanley. 2002. "Analysis of Symbolic Sequences Using the Jensen-Shannon Divergence." *Physical Review E* 65 (4): 041905.

Kullback, Solomon. 1951. "Kullback-Leibler Divergence."

Li, Wenkai, Qinghua Guo, Marek Jakubowski, and Maggi Kelly. 2012. "A New Method for Segmenting Individual Trees from the Lidar Point Cloud." *Photogrammetric Engineering and Remote Sensing* 78 (January): 75–84. https://doi.org/10.14358/PERS.78.1.75.

Popescu, Sorin, and Randolph Wynne. 2004. "Seeing the Trees in the Forest: Using Lidar and Multispectral Data Fusion with Local Filtering and Variable Window Size for Estimating Tree Height." *Photogrammetric Engineering and Remote Sensing* 70 (May): 589–604. https://doi.org/10.14358/PERS.70.5.589.

Welle, Torsten, Lukas Aschenbrenner, Kevin Kuonath, Stefan Kirmaier, and Jonas Franke. 2022. "Mapping Dominant Tree Species of German Forests." *Remote Sensing* 14 (14). https://doi.org/10.3390/rs14143330.

5 Appendix

5.1 Script which can be used to do all preprocessing

Load the file with the research areas ::: {.cell}

```
sf <- sf::read_sf(here::here("research_areas.shp"))
print(sf)
```

```
Simple feature collection with 12 features and 3 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 7.071625 ymin: 51.0895 xmax: 8.539877 ymax: 52.25983
Geodetic CRS:  WGS 84
# A tibble: 12 x 4
      id species name                                                  geometry
   <dbl> <chr>   <chr>                                              <POLYGON [°]>
 1     1 oak     rinkerode          ((7.678922 51.85789, 7.675446 51.85752, 7.~
 2     2 oak     hamm               ((7.858955 51.66699, 7.866444 51.66462, 7.~
 3     3 oak     muenster           ((7.618908 51.9154, 7.617384 51.9172, 7.61~
 4     4 pine    greffen            ((8.168691 51.98965, 8.167178 51.99075, 8.~
 5     5 pine    telgte             ((7.779728 52.00662, 7.781616 52.00662, 7.~
 6     6 pine    mesum              ((7.534424 52.25499, 7.53378 52.25983, 7.5~
 7     7 beech   bielefeld_brackwede ((8.524749 51.9921, 8.528418 51.99079, 8.5~
 8     8 beech   wuelfenrath        ((7.071625 51.29256, 7.072311 51.29334, 7.~
```

```
 9     9 beech   billerbeck          ((7.324729 51.99783, 7.323548 51.99923, 7.~
10    11 spruce  brilon              ((8.532195 51.41029, 8.535027 51.41064, 8.~
11    12 spruce  osterwald           ((8.369328 51.21693, 8.371238 51.21718, 8.~
12    10 spruce  oberhundem          ((8.18082 51.08999, 8.180868 51.09143, 8.1~
```

:::

Init the project ::: {.cell}

```
library(lfa)
sf::sf_use_s2(FALSE)
locations <- lfa_init("research_areas.shp")
```

:::

Do all of the prprocessing steps ::: {.cell}

```
lfa_map_tile_locations(locations,retile,check_flag = "retile")
```

```
No further processing: flag retile is set!Function is already computed, no further computings
```

```
NULL
```

```
lfa_map_tile_locations(locations, lfa_intersect_areas, ctg = NULL, areas_sf = sf,check_fla
```

```
No further processing: flag intersect is set!Function is already computed, no further comput
```

```
NULL
```

```
lfa_map_tile_locations(locations, lfa_ground_correction, ctg = NULL,check_flag = "z_correc
```

```
No further processing: flag z_correction is set!Function is already computed, no further com
```

```
NULL
```

```
lfa_map_tile_locations(locations, lfa_segmentation, ctg = NULL,check_flag = "segmentation"
```

```
No further processing: flag segmentation is set!Function is already computed, no further comp

NULL
```

```
lfa_map_tile_locations(locations, lfa_detection, catalog = NULL, write_to_file = TRUE,chec
```

```
No further processing: flag detection is set!Function is already computed, no further computi

NULL
```

:::

5.2 Quantitative Results

5.2.1 Distribution of Z-Values

```
data <- lfa::lfa_get_detections()
value_column <- "Z"
```

Kullback-Leibler-Divergence

```
kld_results_specie <- lfa::lfa_run_test_asymmetric(data,value_column,"specie",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_specie,"Kullback-Leibler-Divergence betwe
```

Table 2: Kullback-Leibler-Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute z-values

	Beech	Oak	Pine	Spruce
Beech	0.0	13.2	12.5	0.76
Oak	4.2	0.0	3.4	5.02
Pine	2.3	5.6	0.0	3.95
Spruce	2.4	14.7	16.1	0.00

```
colMeans(kld_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 5.252696
```

```
specie <- data[data$specie=="beech",]
kld_results_beech <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_
lfa::lfa_generate_result_table_tests(kld_results_beech,"Kullback-Leibler-Divergence betwee
```

Table 3: Kullback-Leibler-Divergence between the researched areas which have the dominante specie beech for the atrribute z-values

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0.00	0.4	3.1
Billerbeck	0.27	0.0	6.0
Wuelfenrath	1.13	2.4	0.0

```
colMeans(kld_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 1.473353
```

```
specie <- data[data$specie=="oak",]
kld_results_oak <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_fr
lfa::lfa_generate_result_table_tests(kld_results_oak,"Kullback-Leibler-Divergence between
```

Table 4: Kullback-Leibler-Divergence between the researched areas which have the dominante specie oak for the atrribute z-values

	Hamm	Muenster	Rinkerode
Hamm	0.0	2.1	16
Muenster	0.4	0.0	17
Rinkerode	7.6	17.8	0

```
colMeans(kld_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 6.779863
```

```
specie <- data[data$specie=="pine",]
kld_results_pine <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_f
lfa::lfa_generate_result_table_tests(kld_results_pine,"Kullback-Leibler-Divergence between
```

Table 5: Kullback-Leibler-Divergence between the researched areas which have the dominante specie pine for the atrribute z-values

	Greffen	Mesum	Telgte
Greffen	0.00	0.74	16
Mesum	0.43	0.00	18
Telgte	3.87	6.82	0

```
colMeans(kld_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 5.129383
```

```
specie <- data[data$specie=="spruce",]
kld_results_spruce <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_spruce,"Kullback-Leibler-Divergence betwe
```

Table 6: Kullback-Leibler-Divergence between the researched areas which have the dominante specie spruce for the atrribute z-values

	Brilon	Oberhundem	Osterwald
Brilon	0.000	0.092	1.7
Oberhundem	0.081	0.000	2.1
Osterwald	1.521	2.178	0.0

```
colMeans(kld_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.8509258
```

Jensen-Shannon Divergence

```
jsd_results_specie <- lfa::lfa_run_test_symmetric(data,value_column,"specie",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_specie,"Jensen-Shannon Divergence between
```

Table 7: Jensen-Shannon Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute z-values

	Beech	Oak	Pine	Spruce
Beech	0	4.5	4.6	2.4
Oak	NA	0.0	3.9	6.1
Pine	NA	NA	0.0	7.1
Spruce	NA	NA	NA	0.0

```
colMeans(jsd_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 2.246663
```

```
specie <- data[data$specie=="beech",]
jsd_results_beech <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_f
lfa::lfa_generate_result_table_tests(jsd_results_beech,"Jensen-Shannon Divergence between
```

Table 8: Jensen-Shannon Divergence between the researched areas which have the dominante specie beech for the atrribute z-values

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0	1.1	3.3
Billerbeck	NA	0.0	4.9
Wuelfenrath	NA	NA	0.0

```
colMeans(jsd_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 1.10555
```

```
specie <- data[data$specie=="oak",]
jsd_results_oak <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fro
lfa::lfa_generate_result_table_tests(jsd_results_oak,"Jensen-Shannon Divergence between ar
```

Table 9: Jensen-Shannon Divergence between the researched areas which have the dominante specie oak for the atrribute z-values

	Hamm	Muenster	Rinkerode
Hamm	0	1.6	6.5
Muenster	NA	0.0	6.4
Rinkerode	NA	NA	0.0

```
colMeans(jsd_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 1.692942
```

```
specie <- data[data$specie=="pine",]
jsd_results_pine <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fr
lfa::lfa_generate_result_table_tests(jsd_results_pine,"Jensen-Shannon Divergence between a
```

Table 10: Jensen-Shannon Divergence between the researched areas which have the dominante specie pine for the atrribute z-values

	Greffen	Mesum	Telgte
Greffen	0	3.1	12
Mesum	NA	0.0	10
Telgte	NA	NA	0

```
colMeans(jsd_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 2.956354
```

```
specie <- data[data$specie=="spruce",]
jsd_results_spruce <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_spruce,"Jensen-Shannon Divergence between
```

Table 11: Jensen-Shannon Divergence between the researched areas which have the dominante specie spruce for the atrribute z-values

	Brilon	Oberhundem	Osterwald
Brilon	0	0.31	4.0
Oberhundem	NA	0.00	5.5
Osterwald	NA	NA	0.0

```
colMeans(jsd_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 1.100383
```

5.2.2 Nearest Neighbours

Distribution of nearest neighbor distances

```
data <- lfa::lfa_combine_sf_obj(lfa::lfa_get_neighbor_paths(),lfa::lfa_get_all_areas())
```

```
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/bielefeld_brac
  using driver `GPKG'
Simple feature collection with 1443 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 466999.8 ymin: 5759839 xmax: 467617.1 ymax: 5760261
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/billerbeck/neig
  using driver `GPKG'
Simple feature collection with 1732 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 384890.8 ymin: 5761918 xmax: 385590.9 ymax: 5762478
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/wuelfenrath/nei
  using driver `GPKG'
Simple feature collection with 2779 features and 102 fields
```

```
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 365546.3 ymin: 5683711 xmax: 366356.1 ymax: 5684321
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/hamm/neighbours.g
  using driver `GPKG'
Simple feature collection with 2441 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 420953.3 ymin: 5723884 xmax: 421596 ymax: 5724609
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/muenster/neighbou
  using driver `GPKG'
Simple feature collection with 1270 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 404615.6 ymin: 5752535 xmax: 405396.8 ymax: 5752971
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/rinkerode/neighbo
  using driver `GPKG'
Simple feature collection with 1643 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 408428.2 ymin: 5746021 xmax: 409014.8 ymax: 5746511
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/greffen/neighbou
  using driver `GPKG'
Simple feature collection with 513 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 442816.1 ymin: 5760217 xmax: 443148.9 ymax: 5760567
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/mesum/neighbours
  using driver `GPKG'
Simple feature collection with 5031 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 399930.6 ymin: 5790412 xmax: 400969.7 ymax: 5790950
```

```
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/telgte/neighbou
  using driver `GPKG'
Simple feature collection with 3368 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 416135.1 ymin: 5761663 xmax: 416697.1 ymax: 5762477
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/brilon/neighb
  using driver `GPKG'
Simple feature collection with 3342 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 467305.7 ymin: 5695055 xmax: 467996.9 ymax: 5695593
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/oberhundem/nei
  using driver `GPKG'
Simple feature collection with 2471 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 442631.7 ymin: 5660096 xmax: 443309.5 ymax: 5660502
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/osterwald/neig
  using driver `GPKG'
Simple feature collection with 2806 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 455822 ymin: 5673761 xmax: 456483.2 ymax: 5674162
Projected CRS: ETRS89 / UTM zone 32N
```

```
value_column <- "Neighbor_1"
```

Kullback-Leibler-Divergence

```
kld_results_specie <- lfa::lfa_run_test_asymmetric(data,value_column,"specie",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_specie,"Kullback-Leibler-Divergence betwe
```

Table 12: Kullback-Leibler-Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute nearest-neighbor-1

	Beech	Oak	Pine	Spruce
Beech	0.000	0.029	0.40	3.3
Oak	0.031	0.000	0.25	3.9
Pine	0.213	0.128	0.00	4.9
Spruce	2.735	3.199	4.52	0.0

```
colMeans(kld_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 1.477983
```

```
specie <- data[data$specie=="beech",]
kld_results_beech <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_
lfa::lfa_generate_result_table_tests(kld_results_beech,"Kullback-Leibler-Divergence betwee
```

Table 13: Kullback-Leibler-Divergence between the researched areas which have the dominante specie beech for the atrribute nearest-neighbor-1

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0.000	0.35	0.051
Billerbeck	0.380	0.00	0.138
Wuelfenrath	0.059	0.15	0.000

```
colMeans(kld_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.1249588
```

```
specie <- data[data$specie=="oak",]
kld_results_oak <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_fr
lfa::lfa_generate_result_table_tests(kld_results_oak,"Kullback-Leibler-Divergence between
```

Table 14: Kullback-Leibler-Divergence between the researched areas which have the dominante specie oak for the atrribute nearest-neighbor-1

	Hamm	Muenster	Rinkerode
Hamm	0.000	0.079	0.078
Muenster	0.092	0.000	0.019
Rinkerode	0.086	0.020	0.000

```
colMeans(kld_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.04167636
```

```
specie <- data[data$specie=="pine",]
kld_results_pine <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_f
lfa::lfa_generate_result_table_tests(kld_results_pine,"Kullback-Leibler-Divergence between
```

Table 15: Kullback-Leibler-Divergence between the researched areas which have the dominante specie pine for the atrribute nearest-neighbor-1

	Greffen	Mesum	Telgte
Greffen	0.00	0.495	0.258
Mesum	0.48	0.000	0.098
Telgte	0.22	0.076	0.000

```
colMeans(kld_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.1812239
```

```
specie <- data[data$specie=="spruce",]
kld_results_spruce <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_spruce,"Kullback-Leibler-Divergence betwe
```

Table 16: Kullback-Leibler-Divergence between the researched areas which have the dominante specie spruce for the atrribute nearest-neighbor-1

	Brilon	Oberhundem	Osterwald
Brilon	0.00	0.67	5.1
Oberhundem	0.41	0.00	7.2
Osterwald	6.09	6.23	0.0

```
colMeans(kld_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 2.863587
```

Jensen-Shannon Divergence

```
jsd_results_specie <- lfa::lfa_run_test_symmetric(data,value_column,"specie",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_specie,"Jensen-Shannon Divergence between
```

Table 17: Jensen-Shannon Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute nearest-neighbor-1

	Beech	Oak	Pine	Spruce
Beech	0	0.22	2.1	9.3
Oak	NA	0.00	1.3	10.6
Pine	NA	NA	0.0	14.7
Spruce	NA	NA	NA	0.0

```
colMeans(jsd_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 2.470051
```

```
specie <- data[data$specie=="beech",]
jsd_results_beech <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_f
lfa::lfa_generate_result_table_tests(jsd_results_beech,"Jensen-Shannon Divergence between
```

Table 18: Jensen-Shannon Divergence between the researched areas which have the dominante specie beech for the atrribute nearest-neighbor-1

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0	2.2	0.39
Billerbeck	NA	0.0	0.85
Wuelfenrath	NA	NA	0.00

```
colMeans(jsd_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.5042359
```

```
specie <- data[data$specie=="oak",]
jsd_results_oak <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fro
lfa::lfa_generate_result_table_tests(jsd_results_oak,"Jensen-Shannon Divergence between ar
```

Table 19: Jensen-Shannon Divergence between the researched areas which have the dominante specie oak for the atrribute nearest-neighbor-1

	Hamm	Muenster	Rinkerode
Hamm	0	0.57	0.61
Muenster	NA	0.00	0.17
Rinkerode	NA	NA	0.00

```
colMeans(jsd_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.1803836
```

```
specie <- data[data$specie=="pine",]
jsd_results_pine <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fr
lfa::lfa_generate_result_table_tests(jsd_results_pine,"Jensen-Shannon Divergence between a
```

Table 20: Jensen-Shannon Divergence between the researched areas which have the dominante specie pine for the atrribute nearest-neighbor-1

	Greffen	Mesum	Telgte
Greffen	0	3.6	1.89
Mesum	NA	0.0	0.68
Telgte	NA	NA	0.00

```
colMeans(jsd_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.891592
```

```
specie <- data[data$specie=="spruce",]
jsd_results_spruce <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_spruce,"Jensen-Shannon Divergence between
```

Table 21: Jensen-Shannon Divergence between the researched areas which have the dominante specie spruce for the atrribute nearest-neighbor-1

	Brilon	Oberhundem	Osterwald
Brilon	0	4.1	16
Oberhundem	NA	0.0	18
Osterwald	NA	NA	0

```
colMeans(jsd_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 4.471632
```

Distribution of distances to 100th nearest neighbor

```
data <- lfa::lfa_combine_sf_obj(lfa::lfa_get_neighbor_paths(),lfa::lfa_get_all_areas())
```

```
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/bielefeld_brac
  using driver `GPKG'
Simple feature collection with 1443 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 466999.8 ymin: 5759839 xmax: 467617.1 ymax: 5760261
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/billerbeck/neig
  using driver `GPKG'
Simple feature collection with 1732 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 384890.8 ymin: 5761918 xmax: 385590.9 ymax: 5762478
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/wuelfenrath/nei
  using driver `GPKG'
Simple feature collection with 2779 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 365546.3 ymin: 5683711 xmax: 366356.1 ymax: 5684321
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/hamm/neighbours.g
  using driver `GPKG'
Simple feature collection with 2441 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 420953.3 ymin: 5723884 xmax: 421596 ymax: 5724609
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/muenster/neighbou
  using driver `GPKG'
Simple feature collection with 1270 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 404615.6 ymin: 5752535 xmax: 405396.8 ymax: 5752971
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/rinkerode/neighbo
  using driver `GPKG'
```

```
Simple feature collection with 1643 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 408428.2 ymin: 5746021 xmax: 409014.8 ymax: 5746511
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/greffen/neighbou
  using driver `GPKG'
Simple feature collection with 513 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 442816.1 ymin: 5760217 xmax: 443148.9 ymax: 5760567
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/mesum/neighbours
  using driver `GPKG'
Simple feature collection with 5031 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 399930.6 ymin: 5790412 xmax: 400969.7 ymax: 5790950
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/telgte/neighbour
  using driver `GPKG'
Simple feature collection with 3368 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 416135.1 ymin: 5761663 xmax: 416697.1 ymax: 5762477
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/brilon/neighbo
  using driver `GPKG'
Simple feature collection with 3342 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 467305.7 ymin: 5695055 xmax: 467996.9 ymax: 5695593
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/oberhundem/nei
  using driver `GPKG'
Simple feature collection with 2471 features and 102 fields
Geometry type: POINT
Dimension:     XY
```

```
Bounding box:  xmin: 442631.7 ymin: 5660096 xmax: 443309.5 ymax: 5660502
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/osterwald/neig
  using driver `GPKG'
Simple feature collection with 2806 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 455822 ymin: 5673761 xmax: 456483.2 ymax: 5674162
Projected CRS: ETRS89 / UTM zone 32N
```

```
value_column <- "Neighbor_100"
```

Kullback-Leibler-Divergence

```
kld_results_specie <- lfa::lfa_run_test_asymmetric(data,value_column,"specie",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_specie,"Kullback-Leibler-Divergence betwe
```

Table 22: Kullback-Leibler-Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute nearest-neighbor-100

	Beech	Oak	Pine	Spruce
Beech	0.000	0.194	0.082	0.89
Oak	0.183	0.000	0.063	0.67
Pine	0.084	0.069	0.000	0.86
Spruce	1.083	0.809	1.200	0.00

```
colMeans(kld_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 0.3862841
```

```
specie <- data[data$specie=="beech",]
kld_results_beech <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_
lfa::lfa_generate_result_table_tests(kld_results_beech,"Kullback-Leibler-Divergence betwee
```

Table 23: Kullback-Leibler-Divergence between the researched areas which have the dominante specie beech for the atrribute nearest-neighbor-100

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0.00	0.12	0.12
Billerbeck	0.14	0.00	0.40
Wuelfenrath	0.12	0.31	0.00

```
colMeans(kld_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.1338066
```

```
specie <- data[data$specie=="oak",]
kld_results_oak <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_fr
lfa::lfa_generate_result_table_tests(kld_results_oak,"Kullback-Leibler-Divergence between
```

Table 24: Kullback-Leibler-Divergence between the researched areas which have the dominante specie oak for the atrribute nearest-neighbor-100

	Hamm	Muenster	Rinkerode
Hamm	0.00	0.19	0.11
Muenster	0.20	0.00	0.06
Rinkerode	0.11	0.07	0.00

```
colMeans(kld_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.08182597
```

```
specie <- data[data$specie=="pine",]
kld_results_pine <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_f
lfa::lfa_generate_result_table_tests(kld_results_pine,"Kullback-Leibler-Divergence between
```

Table 25: Kullback-Leibler-Divergence between the researched areas which have the dominante specie pine for the atrribute nearest-neighbor-100

	Greffen	Mesum	Telgte
Greffen	0.00	0.25	0.51
Mesum	0.20	0.00	0.25
Telgte	0.54	0.26	0.00

```
colMeans(kld_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.22229
```

```
specie <- data[data$specie=="spruce",]
kld_results_spruce <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_spruce,"Kullback-Leibler-Divergence betwe
```

Table 26: Kullback-Leibler-Divergence between the researched areas which have the dominante specie spruce for the atrribute nearest-neighbor-100

	Brilon	Oberhundem	Osterwald
Brilon	0.000	0.05	0.23
Oberhundem	0.046	0.00	0.37
Osterwald	0.276	0.46	0.00

```
colMeans(kld_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.1591879
```

Jensen-Shannon Divergence

```
jsd_results_specie <- lfa::lfa_run_test_symmetric(data,value_column,"specie",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_specie,"Jensen-Shannon Divergence between
```

Table 27: Jensen-Shannon Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute nearest-neighbor-100

	Beech	Oak	Pine	Spruce
Beech	0	0.38	0.14	1.27
Oak	NA	0.00	0.30	0.78
Pine	NA	NA	0.00	1.39
Spruce	NA	NA	NA	0.00

```
colMeans(jsd_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 0.2997233
```

```
specie <- data[data$specie=="beech",]
jsd_results_beech <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_f
lfa::lfa_generate_result_table_tests(jsd_results_beech,"Jensen-Shannon Divergence between
```

Table 28: Jensen-Shannon Divergence between the researched areas which have the dominante specie beech for the atrribute nearest-neighbor-100

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0	0.22	0.21
Billerbeck	NA	0.00	0.57
Wuelfenrath	NA	NA	0.00

```
colMeans(jsd_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.124106
```

```
specie <- data[data$specie=="oak",]
jsd_results_oak <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fro
lfa::lfa_generate_result_table_tests(jsd_results_oak,"Jensen-Shannon Divergence between ar
```

Table 29: Jensen-Shannon Divergence between the researched areas which have the dominante specie oak for the atrribute nearest-neighbor-100

	Hamm	Muenster	Rinkerode
Hamm	0	0.34	0.17
Muenster	NA	0.00	0.23
Rinkerode	NA	NA	0.00

```
colMeans(jsd_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.1007612
```

```
specie <- data[data$specie=="pine",]
jsd_results_pine <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fr
lfa::lfa_generate_result_table_tests(jsd_results_pine,"Jensen-Shannon Divergence between a
```

Table 30: Jensen-Shannon Divergence between the researched areas which have the dominante specie pine for the atrribute nearest-neighbor-100

	Greffen	Mesum	Telgte
Greffen	0	0.45	0.86
Mesum	NA	0.00	0.50
Telgte	NA	NA	0.00

```
colMeans(jsd_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.2265055
```

```
specie <- data[data$specie=="spruce",]
jsd_results_spruce <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_spruce,"Jensen-Shannon Divergence between
```

Table 31: Jensen-Shannon Divergence between the researched areas which have the dominante specie spruce for the atrribute nearest-neighbor-100

	Brilon	Oberhundem	Osterwald
Brilon	0	0.1	0.57
Oberhundem	NA	0.0	0.73
Osterwald	NA	NA	0.00

```
colMeans(jsd_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.1613747
```

Distribution of average nearest neighbor distances

```
data <- lfa::lfa_combine_sf_obj(lfa::lfa_get_neighbor_paths(),lfa::lfa_get_all_areas())
```

```
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/bielefeld_brac
  using driver `GPKG'
Simple feature collection with 1443 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 466999.8 ymin: 5759839 xmax: 467617.1 ymax: 5760261
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/billerbeck/neig
  using driver `GPKG'
Simple feature collection with 1732 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 384890.8 ymin: 5761918 xmax: 385590.9 ymax: 5762478
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/beech/wuelfenrath/nei
  using driver `GPKG'
Simple feature collection with 2779 features and 102 fields
Geometry type: POINT
Dimension:     XY
```

```
Bounding box:  xmin: 365546.3 ymin: 5683711 xmax: 366356.1 ymax: 5684321
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/hamm/neighbours.g
  using driver `GPKG'
Simple feature collection with 2441 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 420953.3 ymin: 5723884 xmax: 421596 ymax: 5724609
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/muenster/neighbou
  using driver `GPKG'
Simple feature collection with 1270 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 404615.6 ymin: 5752535 xmax: 405396.8 ymax: 5752971
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/oak/rinkerode/neighbo
  using driver `GPKG'
Simple feature collection with 1643 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 408428.2 ymin: 5746021 xmax: 409014.8 ymax: 5746511
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/greffen/neighbou
  using driver `GPKG'
Simple feature collection with 513 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 442816.1 ymin: 5760217 xmax: 443148.9 ymax: 5760567
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/mesum/neighbours
  using driver `GPKG'
Simple feature collection with 5031 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 399930.6 ymin: 5790412 xmax: 400969.7 ymax: 5790950
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
```

```
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/pine/telgte/neighbou
  using driver `GPKG'
Simple feature collection with 3368 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 416135.1 ymin: 5761663 xmax: 416697.1 ymax: 5762477
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/brilon/neighb
  using driver `GPKG'
Simple feature collection with 3342 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 467305.7 ymin: 5695055 xmax: 467996.9 ymax: 5695593
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/oberhundem/nei
  using driver `GPKG'
Simple feature collection with 2471 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 442631.7 ymin: 5660096 xmax: 443309.5 ymax: 5660502
Projected CRS: ETRS89 / UTM zone 32N
Reading layer `neighbours' from data source
  `/home/jakob/gi-master/project-courses/lidar-forest-analysis/src/data/spruce/osterwald/neig
  using driver `GPKG'
Simple feature collection with 2806 features and 102 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 455822 ymin: 5673761 xmax: 456483.2 ymax: 5674162
Projected CRS: ETRS89 / UTM zone 32N
```

```r
names <- paste0("Neighbor_",1:100)
data$avg = rowMeans(dplyr::select(as.data.frame(data),names))
value_column <- "avg"
```

Kullback-Leibler-Divergence

```r
kld_results_specie <- lfa::lfa_run_test_asymmetric(data,value_column,"specie",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_specie,"Kullback-Leibler-Divergence betwe
```

Table 32: Kullback-Leibler-Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute nearest-neighbor-avg

	Beech	Oak	Pine	Spruce
Beech	0.000	0.31	0.065	1.28
Oak	0.302	0.00	0.178	0.83
Pine	0.067	0.17	0.000	1.23
Spruce	1.660	0.92	1.869	0.00

```
colMeans(kld_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 0.5552882
```

```
specie <- data[data$specie=="beech",]
kld_results_beech <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_
lfa::lfa_generate_result_table_tests(kld_results_beech,"Kullback-Leibler-Divergence betwee
```

Table 33: Kullback-Leibler-Divergence between the researched areas which have the dominante specie beech for the atrribute nearest-neighbor-avg

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0.000	0.052	0.50
Billerbeck	0.052	0.000	0.91
Wuelfenrath	0.348	0.612	0.00

```
colMeans(kld_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.27574
```

```
specie <- data[data$specie=="oak",]
kld_results_oak <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_fr
lfa::lfa_generate_result_table_tests(kld_results_oak,"Kullback-Leibler-Divergence between
```

Table 34: Kullback-Leibler-Divergence between the researched areas which have the dominante specie oak for the atrribute nearest-neighbor-avg

	Hamm	Muenster	Rinkerode
Hamm	0.00	0.166	0.217
Muenster	0.16	0.000	0.031
Rinkerode	0.21	0.037	0.000

```
colMeans(kld_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.09154318
```

```
specie <- data[data$specie=="pine",]
kld_results_pine <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_f
lfa::lfa_generate_result_table_tests(kld_results_pine,"Kullback-Leibler-Divergence between
```

Table 35: Kullback-Leibler-Divergence between the researched areas which have the dominante specie pine for the atrribute nearest-neighbor-avg

	Greffen	Mesum	Telgte
Greffen	0.00	0.17	0.29
Mesum	0.14	0.00	0.30
Telgte	0.26	0.32	0.00

```
colMeans(kld_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.1637513
```

```
specie <- data[data$specie=="spruce",]
kld_results_spruce <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_spruce,"Kullback-Leibler-Divergence betwe
```

Table 36: Kullback-Leibler-Divergence between the researched areas which have the dominante specie spruce for the atrribute nearest-neighbor-avg

	Brilon	Oberhundem	Osterwald
Brilon	0.000	0.11	0.29
Oberhundem	0.097	0.00	0.59
Osterwald	0.341	0.75	0.00

```
colMeans(kld_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.2404004
```

Jensen-Shannon Divergence

```
jsd_results_specie <- lfa::lfa_run_test_symmetric(data,value_column,"specie",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_specie,"Jensen-Shannon Divergence between
```

Table 37: Jensen-Shannon Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute nearest-neighbor-avg

	Beech	Oak	Pine	Spruce
Beech	0	0.73	0.19	2.6
Oak	NA	0.00	0.64	1.4
Pine	NA	NA	0.00	3.0
Spruce	NA	NA	NA	0.0

```
colMeans(jsd_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 0.5999417
```

```
specie <- data[data$specie=="beech",]
jsd_results_beech <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_f
lfa::lfa_generate_result_table_tests(jsd_results_beech,"Jensen-Shannon Divergence between
```

Table 38: Jensen-Shannon Divergence between the researched areas which have the dominante specie beech for the atrribute nearest-neighbor-avg

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0	0.14	1.0
Billerbeck	NA	0.00	1.7
Wuelfenrath	NA	NA	0.0

```
colMeans(jsd_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.3215991
```

```
specie <- data[data$specie=="oak",]
jsd_results_oak <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fro
lfa::lfa_generate_result_table_tests(jsd_results_oak,"Jensen-Shannon Divergence between ar
```

Table 39: Jensen-Shannon Divergence between the researched areas which have the dominante specie oak for the atrribute nearest-neighbor-avg

	Hamm	Muenster	Rinkerode
Hamm	0	0.41	0.53
Muenster	NA	0.00	0.26
Rinkerode	NA	NA	0.00

```
colMeans(jsd_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.1558436
```

```
specie <- data[data$specie=="pine",]
jsd_results_pine <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fr
lfa::lfa_generate_result_table_tests(jsd_results_pine,"Jensen-Shannon Divergence between a
```

Table 40: Jensen-Shannon Divergence between the researched areas which have the dominante specie pine for the atrribute nearest-neighbor-avg

	Greffen	Mesum	Telgte
Greffen	0	0.44	0.76
Mesum	NA	0.00	0.89
Telgte	NA	NA	0.00

```
colMeans(jsd_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.2560143
```

```
specie <- data[data$specie=="spruce",]
jsd_results_spruce <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_spruce,"Jensen-Shannon Divergence between
```

Table 41: Jensen-Shannon Divergence between the researched areas which have the dominante specie spruce for the atrribute nearest-neighbor-avg

	Brilon	Oberhundem	Osterwald
Brilon	0	0.32	1.1
Oberhundem	NA	0.00	1.8
Osterwald	NA	NA	0.0

```
colMeans(jsd_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.3713411
```

5.2.3 Distribution of the number of returns

```
data <- sf::st_read("data/tree_properties.gpkg")
neighbors <- lfa::lfa_get_neighbor_paths() |> lfa::lfa_combine_sf_obj(lfa::lfa_get_all_are
data = sf::st_join(data,neighbors, join = sf::st_within)
value_column <- "number_of_returns"
```

Kullback-Leibler-Divergence

```
kld_results_specie <- lfa::lfa_run_test_asymmetric(data,value_column,"specie",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_specie,"Kullback-Leibler-Divergence betwe
```

Table 42: Kullback-Leibler-Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute number-of-returns

	Beech	Oak	Pine	Spruce
Beech	0.000	0.083	0.57	0.049
Oak	0.051	0.000	0.84	0.059
Pine	0.432	0.833	0.00	0.526
Spruce	0.036	0.059	0.54	0.000

```
colMeans(kld_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 0.2550987
```

```
specie <- data[data$specie=="beech",]
kld_results_beech <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_
lfa::lfa_generate_result_table_tests(kld_results_beech,"Kullback-Leibler-Divergence betwee
```

Table 43: Kullback-Leibler-Divergence between the researched areas which have the dominante specie beech for the atrribute number-of-returns

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0.00	0.15	0.082
Billerbeck	0.21	0.00	0.136
Wuelfenrath	0.13	0.19	0.000

```
colMeans(kld_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.09985223
```

```
specie <- data[data$specie=="oak",]
kld_results_oak <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_fr
lfa::lfa_generate_result_table_tests(kld_results_oak,"Kullback-Leibler-Divergence between
```

Table 44: Kullback-Leibler-Divergence between the researched areas which have the dominante specie oak for the atrribute number-of-returns

	Hamm	Muenster	Rinkerode
Hamm	0.00	0.46	0.846
Muenster	0.41	0.00	0.077
Rinkerode	0.81	0.09	0.000

```
colMeans(kld_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.2994815
```

```
specie <- data[data$specie=="pine",]
kld_results_pine <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld_f
lfa::lfa_generate_result_table_tests(kld_results_pine,"Kullback-Leibler-Divergence between
```

Table 45: Kullback-Leibler-Divergence between the researched areas which have the dominante specie pine for the atrribute number-of-returns

	Greffen	Mesum	Telgte
Greffen	0.00	0.1444	0.1773
Mesum	0.14	0.0000	0.0047
Telgte	0.16	0.0045	0.0000

```
colMeans(kld_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.07005788
```

```
specie <- data[data$specie=="spruce",]
kld_results_spruce <- lfa::lfa_run_test_asymmetric(specie,value_column,"area",lfa::lfa_kld
lfa::lfa_generate_result_table_tests(kld_results_spruce,"Kullback-Leibler-Divergence betwe
```

Table 46: Kullback-Leibler-Divergence between the researched areas which have the dominante specie spruce for the atrribute number-of-returns

	Brilon	Oberhundem	Osterwald
Brilon	0.000	0.04	0.034
Oberhundem	0.041	0.00	0.079
Osterwald	0.045	0.10	0.000

```
colMeans(kld_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.03779495
```

Jensen-Shannon Divergence

```
jsd_results_specie <- lfa::lfa_run_test_symmetric(data,value_column,"specie",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_specie,"Jensen-Shannon Divergence between
```

Table 47: Jensen-Shannon Divergence between the researched species Beech, Oak, Pine and Spruce for the atrribute number-of-returns

	Beech	Oak	Pine	Spruce
Beech	0	3e-04	0.019	0.0014
Oak	NA	0e+00	0.021	0.0016
Pine	NA	NA	0.000	0.0143
Spruce	NA	NA	NA	0.0000

```
colMeans(jsd_results_specie, na.rm = TRUE) |> mean()
```

```
[1] 0.004419638
```

```
specie <- data[data$specie=="beech",]
jsd_results_beech <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_f
lfa::lfa_generate_result_table_tests(jsd_results_beech,"Jensen-Shannon Divergence between
```

Table 48: Jensen-Shannon Divergence between the researched areas which have the dominante specie beech for the atrribute number-of-returns

	Bielefeld_brackwede	Billerbeck	Wuelfenrath
Bielefeld_brackwede	0	0.0035	0.00099
Billerbeck	NA	0.0000	0.00554
Wuelfenrath	NA	NA	0.00000

```
colMeans(jsd_results_beech, na.rm = TRUE) |> mean()
```

```
[1] 0.001314268
```

```
specie <- data[data$specie=="oak",]
jsd_results_oak <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fro
lfa::lfa_generate_result_table_tests(jsd_results_oak,"Jensen-Shannon Divergence between ar
```

Table 49: Jensen-Shannon Divergence between the researched areas which have the dominante specie oak for the atrribute number-of-returns

	Hamm	Muenster	Rinkerode
Hamm	0	0.0068	0.0128
Muenster	NA	0.0000	0.0017
Rinkerode	NA	NA	0.0000

```
colMeans(jsd_results_oak, na.rm = TRUE) |> mean()
```

```
[1] 0.002747351
```

```
specie <- data[data$specie=="pine",]
jsd_results_pine <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_fr
lfa::lfa_generate_result_table_tests(jsd_results_pine,"Jensen-Shannon Divergence between a
```

Table 50: Jensen-Shannon Divergence between the researched areas which have the dominante specie pine for the atrribute number-of-returns

	Greffen	Mesum	Telgte
Greffen	0	0.0035	0.00458
Mesum	NA	0.0000	0.00037
Telgte	NA	NA	0.00000

```
colMeans(jsd_results_pine, na.rm = TRUE) |> mean()
```

```
[1] 0.001130537
```

```
specie <- data[data$specie=="spruce",]
jsd_results_spruce <- lfa::lfa_run_test_symmetric(specie,value_column,"area",lfa::lfa_jsd_
lfa::lfa_generate_result_table_tests(jsd_results_spruce,"Jensen-Shannon Divergence between
```

Table 51: Jensen-Shannon Divergence between the researched areas which have the dominante specie spruce for the atrribute number-of-returns

	Brilon	Oberhundem	Osterwald
Brilon	0	0.0069	0.005
Oberhundem	NA	0.0000	0.002
Osterwald	NA	NA	0.000

```
colMeans(jsd_results_spruce, na.rm = TRUE) |> mean()
```

```
[1] 0.001939104
```

5.3 Documentation

5.3.1 `lfa_calculate_patch_density`

Calculate patch density for specified areas based on detection data

Arguments

Argument	Description
`areas_location`	The file path to a shapefile containing spatial polygons representing the areas for which patch density needs to be calculated. Default is "research_areas.shp".
`detections`	A data frame containing detection information, where each row represents a detection and includes the 'area' column specifying the corresponding area. Default is obtained using lfa_get_detections().

Description

This function calculates patch density for specified areas using detection data. It reads the spatial polygons from a shapefile, computes the area size for each patch, counts the number of detections in each patch, and calculates the patch density.

Value

A data frame with patch density information for each specified area. Columns include 'name' (area name), 'geometry' (polygon geometry), 'area_size' (patch area size), 'detections' (number of detections in the patch), and 'density' (computed patch density).

Examples

```
# Assuming you have a shapefile 'your_research_areas.shp' and detection data
# from lfa_get_detections()
density_data <- lfa_calculate_patch_density(areas_location = "your_research_areas.shp")
print(density_data)
```

Usage

```
lfa_calculate_patch_density(
  areas_location = "research_areas.shp",
  detections = lfa::lfa_get_detections()
)
```

5.3.2 `lfa_capitalize_first_char`

Capitalize First Character of a String

Arguments

Argument	Description
`input_string`	A single-character string to be processed.

Concept

String Manipulation

Description

This function takes a string as input and returns the same string with the first character capitalized. If the first character is already capitalized, the function does nothing. If the first character is not from the alphabet, an error is thrown.

Details

This function performs the following steps:

- Checks if the input is a single-character string.
- Verifies if the first character is from the alphabet (A-Z or a-z).
- If the first character is not already capitalized, it capitalizes it.
- Returns the modified string.

Keyword

alphabet

Note

This function is case-sensitive and assumes ASCII characters.

References

None

Seealso

This function is related to the basic string manipulation functions in base R.

Value

A modified string with the first character capitalized if it is not already. If the first character is already capitalized, the original string is returned.

Examples

```
# Capitalize the first character of a string
capitalize_first_char("hello") # Returns "Hello"
capitalize_first_char("World") # Returns "World"

# Error example (non-alphabetic first character)
capitalize_first_char("123abc") # Throws an error
```

Usage

```
lfa_capitalize_first_char(input_string)
```

5.3.3 `lfa_check_flag`

Check if a flag is set, indicating the completion of a specific process.

Arguments

Argument	Description
`flag_name`	A character string specifying the name of the flag file. It should be a descriptive and unique identifier for the process being checked.

Description

This function checks for the existence of a hidden flag file at a specified location within the working directory. If the flag file is found, a message is printed, and the function returns `TRUE` to indicate that the associated processing step has already been completed. If the flag file is not found, the function returns `FALSE` , indicating that further processing can proceed.

Value

A logical value indicating whether the flag is set (`TRUE`) or not (`FALSE`).

Examples

```
# Check if the flag for a process named "data_processing" is set
lfa_check_flag("data_processing")
```

Usage

```
lfa_check_flag(flag_name)
```

5.3.4 `lfa_combine_sf_obj`

Combine Spatial Feature Objects from Multiple GeoPackage Files

Arguments

Argument	Description
`paths`	A character vector containing file paths to GeoPackage files with neighbor information.
`area_infos`	A data frame or list containing information about the corresponding detection areas, including "area" and "specie" columns.

Description

This function reads spatial feature objects (sf) from multiple GeoPackage files and combines them into a single sf object. Each GeoPackage file is assumed to contain neighbor information for a specific detection area, and the resulting sf object includes additional columns indicating the corresponding area and species information.

Value

A combined sf object with additional columns for area and specie information.

Examples

```
# Assuming paths and area_infos are defined
combined_sf <- lfa_combine_sf_obj(paths, area_infos)

# Print the combined sf object
print(combined_sf)
```

Usage

```
lfa_combine_sf_obj(paths, area_infos)
```

5.3.5 lfa_count_returns_all_areas

Count tree returns for all species and areas, returning a consolidated data frame.

Description

This function iterates through all species and areas obtained from the function lfa_get_all_areas . For each combination of species and area, it reads the corresponding area as a catalog, counts the returns per tree using lfa_count_returns_per_tree , and consolidates the results into a data frame. The resulting data frame includes columns for the species, area, and return counts per tree.

Keyword

counting

Seealso

lfa_get_all_areas , lfa_read_area_as_catalog , lfa_count_returns_per_tree

Value

A data frame with columns for species, area, and return counts per tree.

Examples

```
# Count tree returns for all species and areas
returns_counts <- lfa_count_returns_all_areas()
```

Usage

```
lfa_count_returns_all_areas()
```

5.3.6 `lfa_count_returns_per_tree`

Count returns per tree for a given lidR catalog.

Arguments

Argument	Description
`ctg`	A lidR catalog object containing LAS files to be processed.

Description

This function takes a lidR catalog as input and counts the returns per tree. It uses the lidR package to read LAS files from the catalog and performs the counting operation on each tree. The result is a data frame containing the counts of returns for each unique tree ID within the lidR catalog.

Keyword

counting

Seealso

`lidR::readLAS` , `lidR::is.empty` , `base::table` , `dplyr::bind_rows`

Value

A data frame with columns for tree ID and the corresponding count of returns.

Examples

```r
# Count returns per tree for a lidR catalog
ctg <- lfa_read_area_as_catalog("SpeciesA", "Area1")
returns_counts_per_tree <- lfa_count_returns_per_tree(ctg)
```

Usage

```r
lfa_count_returns_per_tree(ctg)
```

5.3.7 `lfa_create_boxplot`

Create a box plot from a data frame

Arguments

Argument	Description
`data`	A data frame containing the data.
`value_column`	The name of the column containing the values for the box plot.
`category_column1`	The name of the column containing the first categorical variable.
`category_column2`	The name of the column containing the second categorical variable.
`title`	An optional title for the plot. If not provided, a default title is generated based on the data frame name.

Description

This function generates a box plot using ggplot2 based on the specified data frame and columns.

Details

The function creates a box plot where the x-axis is based on the second categorical variable, the y-axis is based on the specified value column, and the box plots are colored based on the first categorical variable. The grouping of box plots is done based on the unique values in the second categorical variable.

Value

A ggplot object representing the box plot.

Examples

```
# Assuming you have a data frame 'your_data' with columns 'value', 'category1', and 'categ
create_boxplot(your_data, "value", "category1", "category2")
```

Usage

```
lfa_create_boxplot(
  data,
  value_column,
  category_column1,
  category_column2,
  title = NULL
)
```

5.3.8 `lfa_create_density_plots`

Create density plots for groups in a data frame

Arguments

Argument	Description
`data`	A data frame containing the data.
`value_column`	The name of the column containing the values for the density plot.
`category_column1`	The name of the column containing the categorical variable for grouping.
`category_column2`	The name of the column containing the categorical variable for arranging plots.
`title`	An optional title for the plot. If not provided, a default title is generated based on the data frame name.
`xlims`	Optional limits for the x-axis. Should be a numeric vector with two elements (lower and upper bounds).

Argument	Description
`ylims`	Optional limits for the y-axis. Should be a numeric vector with two elements (lower and upper bounds).

Description

This function generates density plots using ggplot2 based on the specified data frame and columns.

Details

The function creates density plots where the x-axis is based on the specified value column, and the density plots are colored based on the first categorical variable. The arrangement of plots is done based on the unique values in the second categorical variable. The plots are arranged in a 2x2 grid.

Value

A ggplot object representing the density plots arranged in a 2x2 grid.

Examples

```
# Assuming you have a data frame 'your_data' with columns 'value', 'category1', and 'categ
create_density_plots(your_data, "value", "category1", "category2", title = "Density Plots"
```

Usage

```
lfa_create_density_plots(
  data,
  value_column,
  category_column1 = "area",
  category_column2 = "specie",
  title = NULL,
  xlims = NULL,
  ylims = NULL
)
```

5.3.9 `lfa_create_grouped_bar_plot`

Create a barplot using ggplot2

Arguments

Argument	Description
`df`	A data frame containing the relevant columns for the barplot.
`value_column`	The column containing the values to be plotted.
`label_column`	The column used for labeling the bars on the x-axis. Default is "name".
`grouping_column`	The column used for grouping the bars. Default is "species".

Description

This function generates a barplot using ggplot2 based on the specified data frame columns. The barplot displays the values from the specified column, grouped by another column. The grouping can be further differentiated by color if desired.

Value

A ggplot2 barplot.

Examples

```
# Assuming you have a data frame 'your_data_frame' with columns "name", "species", and "va
lfa_create_barplot(your_data_frame, value_column = "value", label_column = "name", groupin
```

Usage

```
lfa_create_grouped_bar_plot(data, grouping_var, value_col, label_col)
```

5.3.10 `lfa_create_neighbor_mean_curves`

Create neighbor mean curves for specified areas

Arguments

Argument	Description
`neighbors`	A data frame containing information about neighbors, where each column represents a specific neighbor, and each row corresponds to an area.
`use_avg`	Logical. If TRUE, the function computes average curves across all neighbors. If FALSE, it computes curves for individual neighbors.

Description

This function generates mean curves for a specified set of areas based on neighbor data. The user can choose to compute mean curves for individual neighbors or averages across neighbors.

Value

A data frame with mean curves for each specified area. Columns represent areas, and rows represent index values.

Examples

```
# Assuming you have a data frame 'your_neighbors_data' with neighbor information
mean_curves <- lfa_create_neighbor_mean_curves(your_neighbors_data, use_avg = TRUE)
print(mean_curves)
```

Usage

```
lfa_create_neighbor_mean_curves(neighbors, use_avg = FALSE)
```

5.3.11 `lfa_create_plot_per_area`

Create a line plot per area with one color per specie

Arguments

Argument	Description
`data`	A data frame with numeric columns and a column named 'specie' for species information.

Description

This function takes a data frame containing numeric columns and creates a line plot using ggplot2. Each line in the plot represents a different area, with one color per specie.

Value

A ggplot2 line plot.

Examples

```
data <- data.frame(
specie = rep(c("Species1", "Species2", "Species3"), each = 10),
column1 = rnorm(30),
column2 = rnorm(30),
column3 = rnorm(30)
)
lfa_create_plot_per_area(data)
```

Usage

```
lfa_create_plot_per_area(data)
```

5.3.12 `lfa_create_stacked_distributions_plot`

Create a stacked distribution plot for tree detections, visualizing the distribution of a specified variable on the x-axis, differentiated by another variable.

Argument	Description

Arguments

Argument	Description
`trees`	A data frame containing tree detection data.
`x_value`	A character string specifying the column name used for finding the values on the x-axis of the histogram.
`fill_value`	A character string specifying the column name by which the data are differentiated in the plot.
`bin`	An integer specifying the number of bins for the histogram. Default is 100.
`ylab`	A character string specifying the y-axis label. Default is "Amount trees."
`xlim`	A numeric vector of length 2 specifying the x-axis limits. Default is c(0, 100).
`ylim`	A numeric vector of length 2 specifying the y-axis limits. Default is c(0, 1000).
`title`	The title of the plot.

Description

This function generates a stacked distribution plot using the ggplot2 package, providing a visual representation of the distribution of a specified variable (`x_value`) on the x-axis, with differentiation based on another variable (`fill_value`). The data for the plot are derived from the provided `trees` data frame.

Keyword

data

Seealso

`ggplot2::geom_histogram` , `ggplot2::facet_wrap` , `ggplot2::ylab` , `ggplot2::scale_fill_brewer` , `ggplot2::coord_cartesian`

Value

A ggplot object representing the stacked distribution plot.

Examples

```
# Create a stacked distribution plot for variable "Z," differentiated by "area"
trees <- lfa_get_detections()
lfa_create_stacked_distributions_plot(trees, "Z", "area")
```

Usage

```
lfa_create_stacked_distributions_plot(
  trees,
  x_value,
  fill_value,
  bin = 100,
  ylab = "Amount trees",
  xlim = c(0, 100),
  ylim = c(0, 1000),
  title =
    "Histograms of height distributions between species 'beech', 'oak', 'pine' and 'spruce
)
```

5.3.13 `lfa_create_stacked_histogram`

Create a stacked histogram for tree detections, summing up the values for each species.

Arguments

Argument	Description
`trees`	A data frame containing tree detection data.
`x_value`	A character string specifying the column name used for finding the values on the x-axis of the histogram.
`fill_value`	A character string specifying the column name by which the data are differentiated in the plot.
`bin`	An integer specifying the number of bins for the histogram. Default is 30.
`ylab`	A character string specifying the y-axis label. Default is "Frequency."
`xlim`	A numeric vector of length 2 specifying the x-axis limits. Default is c(0, 100).

Argument	Description
`ylim`	A numeric vector of length 2 specifying the y-axis limits. Default is NULL.

Description

This function generates a stacked histogram using the ggplot2 package, summing up the values for each species and visualizing the distribution of a specified variable (`x_value`) on the x-axis, differentiated by another variable (`fill_value`). The data for the plot are derived from the provided `trees` data frame.

Keyword

data

Seealso

`ggplot2::geom_histogram` , `ggplot2::ylab` , `ggplot2::scale_fill_brewer` , `ggplot2::coord_cartesian`

Value

A ggplot object representing the stacked histogram.

Examples

```
# Create a stacked histogram for variable "Z," differentiated by "area"
trees <- lfa_get_detections()
lfa_create_stacked_histogram(trees, "Z", "area")
```

Usage

```
lfa_create_stacked_histogram(
  trees,
  x_value,
  fill_value,
  bin = 30,
  ylab = "Frequency",
  xlim = c(0, 100),
  ylim = NULL
```

```
)
```

5.3.14 `lfa_create_tile_location_objects`

Create tile location objects

Author

Jakob Danel

Description

This function traverses a directory structure to find LAZ files and creates tile location objects for each file. The function looks into the the `data` directory of the repository/working directory. It then creates `tile_location` objects based on the folder structure. The folder structure should not be touched by hand, but created by `lfa_init_data_structure()` which builds the structure based on a shape file.

Seealso

`tile_location`

Value

A vector containing tile location objects.

Examples

```
lfa_create_tile_location_objects()

lfa_create_tile_location_objects()
```

Usage

```
lfa_create_tile_location_objects()
```

5.3.15 `lfa_detection`

Perform tree detection on a lidar catalog and optionally save the results to a file.

Arguments

Argument	Description
`catalog`	A lidar catalog containing point cloud data. If set to NULL, the function attempts to read the catalog from the specified tile location.
`tile_location`	An object specifying the location of the lidar tile. If catalog is NULL, the function attempts to read the catalog from this tile location.
`write_to_file`	A logical value indicating whether to save the detected tree information to a file. Default is TRUE.

Description

This function utilizes lidar data to detect trees within a specified catalog. The detected tree information can be optionally saved to a file in the GeoPackage format. The function uses parallel processing to enhance efficiency.

Value

A sf style data frame containing information about the detected trees.

Examples

```
# Perform tree detection on a catalog and save the results to a file
lfa_detection(catalog = my_catalog, tile_location = my_tile_location, write_to_file = TRUE
```

Usage

```
lfa_detection(catalog, tile_location, write_to_file = TRUE)
```

5.3.16 `lfa_download_areas`

Download areas based on spatial features

Arguments

Argument	Description
`sf_areas`	Spatial features representing areas to be downloaded. It must include columns like "species" "name" See details for more information.

Author

Jakob Danel

Description

This function initiates the data structure and downloads areas based on spatial features.

Details

The input data frame, `sf_areas` , must have the following columns:

- "species": The species associated with the area.
- "name": The name of the area.

The function uses the `lfa_init_data_structure` function to set up the data structure and then iterates through the rows of `sf_areas` to download each specified area.

Value

None

Examples

```
lfa_download_areas(sf_areas)


# Example spatial features data frame
sf_areas <- data.frame(
```

```
species = c("SpeciesA", "SpeciesB"),
name = c("Area1", "Area2"),
# Must include also other attributes specialized to sf objects
# such as geometry, for processing of the download
)

lfa_download_areas(sf_areas)
```

Usage

```
lfa_download_areas(sf_areas)
```

5.3.17 `lfa_download`

Download an las file from the state NRW from a specific location

Arguments

Argument	Description
`species`	The species of the tree which is observed at this location
`name`	The name of the area that is observed
`location`	An sf object, which holds the location information for the area where the tile should be downloaded from.

Description

It will download the file and save it to data/ list(list(“html”), list(list(“”))) / list(list(“html”), list(list(“”))) with the name of the tile

Value

The LASCatalog object of the downloaded file

Usage

```
lfa_download(species, name, location)
```

5.3.18 `lfa_find_n_nearest_trees`

Find n Nearest Trees

Arguments

Argument	Description
`trees`	A sf object containing tree coordinates.
`n`	The number of nearest trees to find for each tree (default is 100).

Description

This function calculates the distances to the n nearest trees for each tree in the input dataset.

Value

A data frame with additional columns representing the distances to the n nearest trees.

Examples

```
# Load tree data using lfa_get_detections() (not provided)
tree_data <- lfa_get_detections()

# Filter tree data for a specific species and area
tree_data = tree_data[tree_data$specie == "pine" & tree_data$area == "greffen", ]

# Find the 100 nearest trees for each tree in the filtered dataset
tree_data <- lfa_find_n_nearest_trees(tree_data)
```

Usage

```
lfa_find_n_nearest_trees(trees, n = 100)
```

5.3.19 `lfa_generate_result_table_tests`

Generate Result Table for Tests

Arguments

Argument	Description
`table`	A data frame representing the result table.

Description

This function generates a result table for tests using the knitr::kable function.

Details

This function uses the knitr::kable function to create a formatted table, making it suitable for HTML output. The input table is expected to be a data frame with test results, and the resulting table will have capitalized row and column names with lines between columns and rows.

Value

A formatted table suitable for HTML output with lines between columns and rows.

Examples

```
# Generate a result table for tests
result_table <- data.frame(
Test1 = c(0.05, 0.10, 0.03),
Test2 = c(0.02, 0.08, 0.01),
Test3 = c(0.08, 0.12, 0.05)
)
formatted_table <- lfa_generate_result_table_tests(result_table)
print(formatted_table)
```

Usage

```
lfa_generate_result_table_tests(table, caption = "Table Caption")
```

5.3.20 `lfa_get_all_areas`

Retrieve a data frame containing all species and corresponding areas.

Description

This function scans the "data" directory within the current working directory to obtain a list of species. It then iterates through each species to retrieve the list of areas associated with that species. The resulting data frame contains two columns: "specie" representing the species and "area" representing the corresponding area.

Keyword

data

Seealso

`list.dirs`

Value

A data frame with columns "specie" and "area" containing information about all species and their associated areas.

Examples

```
# Retrieve a data frame with information about all species and areas
all_areas_df <- lfa_get_all_areas()
```

Usage

```
lfa_get_all_areas()
```

5.3.21 `lfa_get_detection_area`

Get Detection for an area

Arguments

Argument	Description
`species`	A character string specifying the target species.
`name`	A character string specifying the name of the tile.

Description

Retrieves the tree detection information for a specified species and tile.

Details

This function reads tree detection data from geopackage files within the specified tile location for a given species. It then combines the data into a single SF data frame and returns it. The function assumes that the tree detection files follow a naming convention with the pattern "_detection.gpkg".

Keyword

spatial

References

This function is part of the LiDAR Forest Analysis (LFA) package.

Seealso

`get_tile_dir`

Value

A Simple Features (SF) data frame containing tree detection information for the specified species and tile.

Examples

```
# Retrieve tree detection data for species "example_species" in tile "example_tile"
trees_data <- lfa_get_detection_tile_location("example_species", "example_tile")

# Example usage:
```

```
trees_data <- lfa_get_detection_tile_location("example_species", "example_tile")

# No trees found scenario:
empty_data <- lfa_get_detection_tile_location("nonexistent_species", "nonexistent_tile")
# The result will be an empty data frame if no trees are found for the specified species a

# Error handling:
# In case of invalid inputs, the function may throw errors. Ensure correct species and til
```

Usage

```
lfa_get_detection_area(species, name)
```

5.3.22 `lfa_get_detections_species`

Retrieve detections for a specific species.

Arguments

Argument	Description
`species`	A character string specifying the target species.

Description

This function retrieves detection data for a given species from multiple areas.

Details

The function looks for detection data in the "data" directory for the specified species. It then iterates through each subdirectory (representing different areas) and consolidates the detection data into a single data frame.

Value

A data frame containing detection information for the specified species in different areas.

Examples

```r
# Example usage:
detections_data <- lfa_get_detections_species("example_species")
```

Usage

```r
lfa_get_detections_species(species)
```

5.3.23 `lfa_get_detections`

Retrieve aggregated detection data for multiple species.

Concept

data retrieval functions

Description

This function obtains aggregated detection data for multiple species by iterating through the list of species obtained from `lfa_get_species` . For each species, it calls `lfa_get_detections_species` to retrieve the corresponding detection data and aggregates the results into a single data frame. The resulting data frame includes columns for the species, tree detection data, and the area in which the detections occurred.

Keyword

aggregation

Seealso

`lfa_get_species` , `lfa_get_detections_species`

Other data retrieval functions: `lfa_get_species`

Value

A data frame containing aggregated detection data for multiple species.

Examples

```
lfa_get_detections()

# Retrieve aggregated detection data for multiple species
detections_data <- lfa_get_detections()
```

Usage

```
lfa_get_detections()
```

5.3.24 `lfa_get_flag_path`

Get the path to a flag file indicating the completion of a specific process.

Arguments

Argument	Description
`flag_name`	A character string specifying the name of the flag file. It should be a descriptive and unique identifier for the process being flagged.

Description

This function constructs and returns the path to a hidden flag file, which serves as an indicator that a particular processing step has been completed. The flag file is created in a designated location within the working directory.

Value

A character string representing the absolute path to the hidden flag file.

Examples

```
# Get the flag path for a process named "data_processing"
lfa_get_flag_path("data_processing")
```

Usage

```
lfa_get_flag_path(flag_name)
```

5.3.25 `lfa_get_neighbor_paths`

Get Paths to Neighbor GeoPackage Files

Description

This function retrieves the file paths to GeoPackage files containing neighbor information for each detection area. The GeoPackage files are assumed to be named "neighbours.gpkg" and organized in a directory structure under the "data" folder.

Value

A character vector containing file paths to GeoPackage files for each detection area's neighbors.

Examples

```
# Get paths to neighbor GeoPackage files for all areas
paths <- lfa_get_neighbor_paths()

# Print the obtained file paths
print(paths)
```

Usage

```
lfa_get_neighbor_paths()
```

5.3.26 `lfa_get_species`

Get a list of species from the data directory.

Concept

data retrieval functions

Description

This function retrieves a list of species by scanning the "data" directory located in the current working directory.

Keyword

data

References

This function relies on the `list.dirs` function for directory listing.

Seealso

`list.dirs`

Other data retrieval functions: `lfa_get_detections`

Value

A character vector containing the names of species found in the "data" directory.

Examples

```
# Retrieve the list of species
species_list <- lfa_get_species()
```

Usage

```
lfa_get_species()
```

5.3.27 `lfa_ground_correction`

Correct the point clouds for correct ground imagery

Arguments

Argument	Description
`ctg`	An LASCatalog object. If not null, it will perform the actions on this object, if NULL inferring the catalog from the tile_location
`tile_location`	A tile_location type object holding the information about the location of the cataog. This is used to save the catalog after processing too.

Author

Jakob Danel

Description

This function is needed to correct the Z value of the point cloud, relative to the real ground height. After using this function to your catalog, the Z values can be seen as the real elevation about the ground. At the moment the function uses the `tin()` function from the `lidr` package. NOTE : The operation is inplace and can not be reverted, the old values of the point cloud will be deleted!

Value

A catalog with the corrected z values. The catalog is always stored at tile_location and holding only the transformed values.

Usage

```
lfa_ground_correction(ctg, tile_location)
```

5.3.28 `lfa_init_data_structure`

Initialize data structure for species and areas

Arguments

Argument	Description
`sf_species`	A data frame with information about species and associated areas.

Description

This function initializes the data structure for storing species and associated areas.

Details

The input data frame, `sf_species` , should have at least the following columns:

- "species": The names of the species for which the data structure needs to be initialized.
- "name": The names of the associated areas.

The function creates directories based on the species and area information provided in the `sf_species` data frame. It checks whether the directories already exist and creates them if they don't.

Value

None

Examples

```
# Example species data frame
sf_species <- data.frame(
species = c("SpeciesA", "SpeciesB"),
name = c("Area1", "Area2"),
# Other necessary columns
)

lfa_init_data_structure(sf_species)

# Example species data frame
sf_species <- data.frame(
species = c("SpeciesA", "SpeciesB"),
name = c("Area1", "Area2"),
# Other necessary columns
)
```

```
lfa_init_data_structure(sf_species)
```

Usage

```
lfa_init_data_structure(sf_species)
```

5.3.29 `lfa_init`

Initialize LFA (LiDAR forest analysis) data processing

Arguments

Argument	Description
`sf_file`	A character string specifying the path to the shapefile containing spatial features of research areas.

Description

This function initializes the LFA data processing by reading a shapefile containing spatial features of research areas, downloading the specified areas, and creating tile location objects for each area.

Details

This function reads a shapefile (`sf_file`) using the `sf` package, which should contain information about research areas. It then calls the `lfa_download_areas` function to download the specified areas and `lfa_create_tile_location_objects` to create tile location objects based on Lidar data files in those areas. The shapefile MUST follow the following requirements:

- Each geometry must be a single object of type polygon
- Each entry must have the following attributes:
- species: A string describing the tree species of the area.
- name: A string describing the location of the area.

Value

A vector containing tile location objects.

Examples

```
# Initialize LFA processing with the default shapefile
lfa_init()

# Initialize LFA processing with a custom shapefile
lfa_init("custom_areas.shp")

# Example usage with the default shapefile
lfa_init()

# Example usage with a custom shapefile
lfa_init("custom_areas.shp")
```

Usage

```
lfa_init(sf_file = "research_areas.shp")
```

5.3.30 `lfa_intersect_areas`

Intersect Lidar Catalog with Spatial Features

Arguments

Argument	Description
`ctg`	A LAScatalog object representing the Lidar data to be processed.
`tile_location`	A tile location object representing the specific area of interest.
`areas_sf`	Spatial features defining areas.

Description

This function intersects a Lidar catalog with a specific area defined by spatial features.

Details

The function intersects the Lidar catalog specified by `ctg` with a specific area defined by the `tile_location` object and `areas_sf` . It removes points outside the specified area and returns a modified LAScatalog object.

The specified area is identified based on the `species` and `name` attributes in the `tile_location` object. If a matching area is not found in `areas_sf` , the function stops with an error.

The function then transforms the spatial reference of the identified area to match that of the Lidar catalog using `sf::st_transform` .

The processing is applied to each chunk in the catalog using the `identify_area` function, which merges spatial information and filters out points that are not classified as inside the identified area. After processing, the function writes the modified LAS files back to the original file locations, removing points outside the specified area.

If an error occurs during the processing of a chunk, a warning is issued, and the function continues processing the next chunks. If no points are found after filtering, a warning is issued, and NULL is returned.

Seealso

Other functions in the Lidar forest analysis (LFA) package.

Value

A modified LAScatalog object with points outside the specified area removed.

Examples

```
# Example usage
lfa_intersect_areas(ctg, tile_location, areas_sf)

# Example usage
lfa_intersect_areas(ctg, tile_location, areas_sf)
```

Usage

```
lfa_intersect_areas(ctg, tile_location, areas_sf)
```

5.3.31 `lfa_jsd_from_vec`

Compute Jensen-Shannon Divergence from Vectors

Arguments

Argument	Description
`x`	A numeric vector.
`y`	A numeric vector.

Description

This function calculates the Jensen-Shannon Divergence (JSD) between two vectors.

Value

Jensen-Shannon Divergence between the density distributions of x and y.

Examples

```
x <- rnorm(100)
y <- rnorm(100, mean = 2)
lfa_jsd_from_vec(x, y)
```

Usage

```
lfa_jsd_from_vec(x, y)
```

5.3.32 `lfa_jsd`

Jensen-Shannon Divergence Calculation

Arguments

Argument	Description
p	A numeric vector representing the probability distribution P.
q	A numeric vector representing the probability distribution Q.
epsilon	A small positive constant added to both P and Q to avoid logarithm of zero. Default is 1e-10.

Description

This function calculates the Jensen-Shannon Divergence (JSD) between two probability distributions P and Q.

Details

The JSD is computed using the Kullback-Leibler Divergence (KLD) as follows: `sum((p * log((p + epsilon) / (m + epsilon)) + q * log((q + epsilon) / (m + epsilon))) / 2)` where `m = (p + q) / 2` .

Seealso

kld , sum , log

Value

A numeric value representing the Jensen-Shannon Divergence between P and Q.

Examples

```
# Calculate JSD between two probability distributions
p_distribution <- c(0.2, 0.3, 0.5)
q_distribution <- c(0.1, 0, 0.9)
jsd_result <- jsd(p_distribution, q_distribution)
print(jsd_result)
```

Usage

```
lfa_jsd(p, q, epsilon = 1e-10)
```

5.3.33 `lfa_kld_from_vec`

Compute Kullback-Leibler Divergence from Vectors

Arguments

Argument	Description
`x`	A numeric vector.
`y`	A numeric vector.

Description

This function calculates the Kullback-Leibler Divergence (KLD) between two vectors.

Value

Kullback-Leibler Divergence between the density distributions of x and y.

Examples

```
x <- rnorm(100)
y <- rnorm(100, mean = 2)
lfa_kld_from_vec(x, y)
```

Usage

```
lfa_kld_from_vec(x, y)
```

5.3.34 `lfa_kld`

Kullback-Leibler Divergence Calculation

Arguments

Argument	Description
p	A numeric vector representing the probability distribution P.
q	A numeric vector representing the probability distribution Q.
epsilon	A small positive constant added to both P and Q to avoid logarithm of zero. Default is 1e-10.

Description

This function calculates the Kullback-Leibler Divergence (KLD) between two probability distributions P and Q.

Details

The KLD is computed using the formula: `sum(p * log((p + epsilon) / (q + epsilon)))` This avoids issues when the denominator (Q) contains zero probabilities.

Seealso

sum , log

Value

A numeric value representing the Kullback-Leibler Divergence between P and Q.

Examples

```
# Calculate KLD between two probability distributions
p_distribution <- c(0.2, 0.3, 0.5)
q_distribution <- c(0.1, 0, 0.9)
kld_result <- kld(p_distribution, q_distribution)
print(kld_result)
```

Usage

```
lfa_kld(p, q, epsilon = 1e-10)
```

5.3.35 `lfa_ks_test`

Kolmogorov-Smirnov Test Wrapper Function

Arguments

Argument	Description
`x`	A numeric vector representing the first sample.
`y`	A numeric vector representing the second sample.
`output_variable`	A character string specifying the output variable to extract from the ks.test result. Default is "p.value". Other possible values include "statistic" and "alternative".
`...`	Additional arguments to be passed to the ks.test function.

Description

This function serves as a wrapper for the Kolmogorov-Smirnov (KS) test between two samples.

Details

The function uses the ks.test function to perform a two-sample KS test and returns the specified output variable. The default output variable is the p-value. Other possible output variables include "statistic" and "alternative".

Seealso

`ks.test`

Value

A numeric value representing the specified output variable from the KS test result.

Examples

```
# Perform KS test and extract the p-value
result <- lfa_ks_test(sample1, sample2)
print(result)
```

```r
# Perform KS test and extract the test statistic
result_statistic <- lfa_ks_test(sample1, sample2, output_variable = "statistic")
print(result_statistic)
```

Usage

```r
lfa_ks_test(x, y, output_variable = "p.value", ...)
```

5.3.36 `lfa_load_ctg_if_not_present`

Loading the catalog if it is not present

Arguments

Argument	Description
`ctg`	Catalog object. Can be NULL
`tile_location`	The location to look for the catalog tiles, if their are not present

Description

This function checks if the catalog is `NULL` . If it is it will load the catalog from the `tile_location`

Value

The provided ctg object if not null, else the catalog for the tiles of the tile_location.

Usage

```r
lfa_load_ctg_if_not_present(ctg, tile_location)
```

5.3.37 `lfa_map_tile_locations`

Map Function Over Tile Locations

Arguments

Argument	Description
`tile_locations`	A list of tile location objects.
`map_function`	The mapping function to be applied to each tile location.
`...`	Additional arguments to be passed to the mapping function.

Description

This function applies a specified mapping function to each tile location in a list.

Details

This function iterates over each tile location in the provided list (`tile_locations`) and applies the specified mapping function (`map_function`) to each tile location. The mapping function should accept a tile location object as its first argument, and additional arguments can be passed using the ellipsis (`...`) syntax.

This function is useful for performing operations on multiple tile locations concurrently, such as loading Lidar data, processing areas, or other tasks that involve tile locations.

Seealso

The mapping function provided should be compatible with the structure and requirements of the tile locations and the specific task being performed.

Value

None

Examples

```
# Example usage
lfa_map_tile_locations(tile_locations, my_mapping_function, param1 = "value")

# Example usage
lfa_map_tile_locations(tile_locations, my_mapping_function, param1 = "value")
```

Usage

```
lfa_map_tile_locations(tile_locations, map_function, check_flag = NULL, ...)
```

5.3.38 `lfa_merge_and_save`

Merge and Save Text Files in a Directory

Arguments

Argument	Description
`input_directory`	The path to the input directory containing text files.
`output_name`	The name for the output file where the merged content will be saved.

Description

This function takes an input directory and an output name as arguments. It merges the textual content of all files in the specified directory into a single string, with each file's content separated by a newline character. The merged content is then saved into a file named after the output name in the same directory. After the merging is complete, all input files are deleted.

Details

This function reads the content of each text file in the specified input directory and concatenates them into a single string. Each file's content is separated by a newline character. The merged content is then saved into a file named after the output name in the same directory. Finally, all input files are deleted from the directory.

Seealso

`readLines` , `writeLines` , `file.remove`

Value

This function does not explicitly return any value. It prints a message indicating the successful completion of the merging and saving process.

Examples

```
# Merge text files in the "data_files" directory and save the result in "merged_output"
lfa_merge_and_save("data_files", "merged_output")

# Merge text files in the "data_files" directory and save the result in "merged_output"
lfa_merge_and_save("data_files", "merged_output")
```

Usage

```
lfa_merge_and_save(input_directory, output_name)
```

5.3.39 `lfa_random_forest`

Random Forest Classifier with Leave-One-Out Cross-Validation

Arguments

Argument	Description
`tree_data`	A data frame containing the tree data, including the response variable ("specie") and predictor variables.
`excluded_input_columns`	A character vector specifying columns to be excluded from predictor variables.
`response_variable`	The response variable to be predicted (default is "specie").
`seed`	An integer to set the seed for reproducibility (default is 123).
`...`	Additional parameters to be passed to the randomForest function.

Description

This function performs a random forest classification using leave-one-out cross-validation for each area in the input tree data. It returns a list containing various results, including predicted species, confusion matrix, accuracy, and the formula used for modeling.

Value

A list containing the following elements:

- `predicted_species_absolute` : A data frame with observed and predicted species for each area.
- `predicted_species_relative` : A data frame wit the relative precictions per speices and areas, normalized by the total predictions in each area.
- `confusion_matrix` : A confusion matrix showing the counts of predicted vs. observed species.
- `accuracy` : The accuracy of the model, calculated as the sum of diagonal elements in the confusion matrix divided by the total count.
- `formula` : The formula used for modeling.

Examples

```
# Assuming tree_data is defined
results <- lfa_random_forest(tree_data, excluded_input_columns = c("column1", "column2"))

# Print the list of results
print(results)
```

Usage

```
lfa_random_forest(
  tree_data,
  excluded_input_columns,
  response_variable = "specie",
  ntree = 100,
  seed = 123,
  ...
)
```

5.3.40 `lfa_rd_to_qmd`

Convert Rd File to Markdown

Arguments

Argument	Description
`rdfile`	The path to the Rd file or a parsed Rd object.
`outfile`	The path to the output Markdown file (including the file extension).
`append`	Logical, indicating whether to append to an existing file (default is FALSE).

Description

IMPORTANT NOTE: This function is nearly identical to the `Rd2md::Rd2markdown` function from the `Rd2md` package. We needed to implement our own version of it because of various reasons:

- The algorithm uses hardcoded header sizes (h1 and h2 in original) which is not feasible for our use-case of the markdown.
- We needed to add some Quarto Markdown specifics, e.g. to make sure that the examples will not be runned.
- We want to exclude certain tags from our implementation.

Details

For that reason we copied the method and made changes as needed and also added this custom documentation.

This function converts an Rd (R documentation) file to Markdown format (.md) and saves the converted file at the specified location. The function allows appending to an existing file or creating a new one. The resulting Markdown file includes sections for the function's name, title, and additional content such as examples, usage, arguments, and other sections present in the Rd file.

The function performs the following steps:

- Parses the Rd file using the Rd2md package.
- Creates a Markdown file with sections for the function's name, title, and additional content.
- Appends the content to an existing file if `append` is set to TRUE.
- Saves the resulting Markdown file at the specified location.

Seealso

Rd2md::parseRd

Value

This function does not explicitly return any value. It saves the converted Markdown file at the specified location as described in the details section.

Examples

```
# Convert Rd file to Markdown and save it
lfa_rd_to_md("path/to/your/file.Rd", "path/to/your/output/file.md")

# Convert Rd file to Markdown and append to an existing file
lfa_rd_to_md("path/to/your/file.Rd", "path/to/existing/output/file.md", append = TRUE)
```

Usage

```
lfa_rd_to_qmd(rdfile, outfile, append = FALSE)
```

5.3.41 `lfa_rd_to_results`

Convert Rd Files to Markdown and Merge Results

Description

This function converts all Rd (R documentation) files in the "man" directory to Markdown format (.qmd) and saves the converted files in the "results/appendix/package-docs" directory. It then merges the converted Markdown files into a single string and saves the merged content into a file named "docs.qmd" in the "results/appendix/package-docs" directory.

Details

The function performs the following steps:

- Removes any existing "docs.qmd" file in the "results/appendix/package-docs" directory.
- Finds all Rd files in the "man" directory.
- Converts each Rd file to Markdown format (.qmd) using the `lfa_rd_to_qmd` function.
- Saves the converted Markdown files in the "results/appendix/package-docs" directory.
- Merges the content of all converted Markdown files into a single string.
- Saves the merged content into a file named "docs.qmd" in the "results/appendix/package-docs" directory.

Seealso

`lfa_rd_to_qmd` , `lfa_merge_and_save`

Value

This function does not explicitly return any value. It performs the conversion, merging, and saving operations as described in the details section.

Examples

```
# Convert Rd files to Markdown and merge the results
lfa_rd_to_results()
```

Usage

```
lfa_rd_to_results()
```

5.3.42 `lfa_read_area_as_catalog`

Read LiDAR data from a specified species and location as a catalog.

Arguments

Argument	Description
`specie`	A character string specifying the species of interest.
`location_name`	A character string specifying the name of the location.

Description

This function constructs the file path based on the specified `specie` and `location_name` , lists the directories at that path, and reads the LiDAR data into a `lidR::LAScatalog` .

Value

A `lidR::LAScatalog` object containing the LiDAR data from the specified location and species.

Examples

```
lfa_read_area_as_catalog("beech", "location1")
```

Usage

```
lfa_read_area_as_catalog(specie, location_name)
```

5.3.43 `lfa_run_test_asymmetric`

Asymmetric Pairwise Test for Categories

Arguments

Argument	Description
`data`	A data frame containing the relevant columns.
`data_column`	A character string specifying the column containing the numerical data.
`category_column`	A character string specifying the column containing the categorical variable.

Argument	Description
test_function	A function used to perform the pairwise test between two sets of data. It should accept two vectors of numeric data and additional parameters specified by ... . The function should return a numeric value representing the test result.
...	Additional parameters to be passed to the test_function .

Description

This function performs an asymmetric pairwise test for categories using a user-defined test_function .

Details

The function calculates the test results for each unique combination of categories using the specified test_function . The resulting table is asymmetric, containing the test results for comparisons from the rows to the columns.

Seealso

outer , Vectorize

Value

A data frame representing the results of the asymmetric pairwise tests between categories.

Examples

```
# Define a custom test function
custom_test_function <- function(x, y) {
# Your test logic here
# Return a numeric result
return(mean(x) - mean(y))
}

# Perform an asymmetric pairwise test
result <- lfa_run_test_asymmetric(your_data, "numeric_column", "category_column", custom_t
```

Usage

```
lfa_run_test_asymmetric(data, data_column, category_column, test_function, ...)
```

5.3.44 `lfa_run_test_symmetric`

Symmetric Pairwise Test for Categories

Arguments

Argument	Description
`data`	A data frame containing the relevant columns.
`data_column`	A character string specifying the column containing the numerical data.
`category_column`	A character string specifying the column containing the categorical variable.
`test_function`	A function used to perform the pairwise test between two sets of data. It should accept two vectors of numeric data and additional parameters specified by `...` . The function should return a numeric value representing the test result.
`...`	Additional parameters to be passed to the `test_function` .

Description

This function performs a symmetric pairwise test for categories using a user-defined `test_function` .

Details

The function calculates the test results for each unique combination of categories using the specified `test_function` . The resulting table is symmetric, containing the test results for comparisons from the rows to the columns. The upper triangle of the matrix is filled with `NA` to avoid duplicate results.

Seealso

`outer` , `Vectorize`

Value

A data frame representing the results of the symmetric pairwise tests between categories.

Examples

```
# Define a custom test function
custom_test_function <- function(x, y) {
# Your test logic here
# Return a numeric result
return(mean(x) - mean(y))
}

# Perform a symmetric pairwise test
result <- lfa_run_test_symmetric(your_data, "numeric_column", "category_column", custom_te
```

Usage

```
lfa_run_test_symmetric(data, data_column, category_column, test_function, ...)
```

5.3.45 `lfa_save_all_neighbours`

Save Neighbors for All Areas

Arguments

Argument	Description
`n`	The number of nearest trees to find for each tree (default is 100).

Description

This function iterates through all detection areas, finds the n nearest trees for each tree, and saves the result to a GeoPackage file for each area.

Examples

```
# Save neighbors for all areas with default value (n=100)
lfa_save_all_neighbours()

# Save neighbors for all areas with a specific value of n (e.g., n=50)
lfa_save_all_neighbours(n = 50)
```

Usage

```
lfa_save_all_neighbours(n = 100)
```

5.3.46 `lfa_segmentation`

Segment the elements of an point cloud by trees

Arguments

Argument	Description
`ctg`	An LASCatalog object. If not null, it will perform the actions on this object, if NULL inferring the catalog from the tile_location
`tile_location`	A tile_location type object holding the information about the location of the catalog. This is used to save the catalog after processing too.

Author

Jakob Danel

Description

This function will try to to divide the hole point cloud into unique trees. Therefore it is assigning for each chunk of the catalog a `treeID` for each point. Therefore the algorithm uses the `li2012` implementation with the following parameters: `li2012(dt1 = 2, dt2 = 3, R = 2, Zu = 10, hmin = 5, speed_up = 12)` NOTE : The operation is in place and can not be reverted, the old values of the point cloud will be deleted!

Value

A catalog where each chunk has additional `treeID` values indicating the belonging tree.

Usage

```
lfa_segmentation(ctg, tile_location)
```

5.3.47 `lfa_set_flag`

Set a flag to indicate the completion of a specific process.

Arguments

Argument	Description
`flag_name`	A character string specifying the name of the flag file. It should be a descriptive and unique identifier for the process being flagged.

Description

This function creates a hidden flag file at a specified location within the working directory to indicate that a particular processing step has been completed. If the flag file already exists, a warning is issued.

Value

This function does not have a formal return value.

Examples

```
# Set the flag for a process named "data_processing"
lfa_set_flag("data_processing")
```

Usage

```
lfa_set_flag(flag_name)
```