Package: tidylearn (via r-universe)

November 2, 2026

Title A Unified Tidy Interface to R's Machine Learning Ecosystem

Version 0.7.0.9000

Description Provides a unified tidyverse-compatible interface to R's machine learning ecosystem - from data ingestion to model publishing. The tl_read() family reads data from files ('CSV', 'Excel', 'Parquet', 'JSON'), databases ('SQLite', 'PostgreSQL', 'MySQL', 'BigQuery'), and cloud sources ('S3', 'GitHub', 'Kaggle'). The tl_model() function wraps established implementations from 'glmnet', 'randomForest', 'xgboost', 'e1071', 'rpart', 'gbm', 'nnet', 'cluster', 'dbscan', and others with consistent function signatures and tidy tibble output. Results flow into unified 'ggplot2'-based visualization and optional formatted 'gt' tables via the tl_table() family. The underlying algorithms are unchanged; 'tidylearn' simply makes them easier to use together. Access raw model objects via the $fit slot for a supervised method, or fitmodel for an unsupervised one. Methods include random forests Breiman (2001) <doi:10.1023/A:1010933404324>, LASSO regression Tibshirani (1996) <doi:10.1111/j.2517-6161.1996.tb02080.x>, elastic net Zou and Hastie (2005) <doi:10.1111/j.1467-9868.2005.00503.x>, support vector machines Cortes and Vapnik (1995) <doi:10.1007/BF00994018>, and gradient boosting Friedman (2001) <doi:10.1214/aos/1013203451>.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports dplyr (>= 1.0.0), ggplot2 (>= 3.4.0), tibble (>= 3.0.0), tidyr (>= 1.0.0), purrr (>= 0.3.0), rlang (>= 1.0.0), magrittr, stats, e1071, gbm, glmnet, nnet, randomForest, rpart, rsample, ROCR, yardstick, cluster (>= 2.1.0), dbscan (>= 1.1.0), MASS, smacof (>= 2.1.0)

Suggests arules, arulesViz, bigrquery, car, DBI, DiagrammeR, DT, GGally, ggforce, gridExtra, gt, httr2 (>= 1.0.0), jsonlite, keras3, knitr, later, lmtest, moments, nanoparquet, NeuralNetTools, paws.storage, readr, RhpcBLASctl, readxl, RMariaDB, rmarkdown, RPostgres, rpart.plot, RSQLite, shiny, shinydashboard, tensorflow, testthat (>= 3.1.7), withr, xgboost

Config/testthat/edition 3

URL https://tidylearn.sheetsolved.com, https://github.com/ces0491/tidylearn

BugReports https://github.com/ces0491/tidylearn/issues

VignetteBuilder knitr

Collate 'tidylearn-package.R' 'utils.R' 'read.R' 'read-backends.R' 'core.R' 'preprocessing.R' 'supervised-classification.R' 'supervised-regression.R' 'supervised-regularization.R' 'supervised-trees.R' 'supervised-svm.R' 'supervised-neural-networks.R' 'supervised-deep-learning.R' 'supervised-xgboost.R' 'unsupervised-distance.R' 'unsupervised-pca.R' 'unsupervised-mds.R' 'unsupervised-clustering.R' 'unsupervised-hclust.R' 'unsupervised-dbscan.R' 'unsupervised-market-basket.R' 'unsupervised-validation.R' 'integration.R' 'pipeline.R' 'model-selection.R' 'compute-detection.R' 'compute-advisor.R' 'compute-routing.R' 'cloud-consent.R' 'cloud-endpoint.R' 'cloud-cost.R' 'cloud-serialize.R' 'tuning.R' 'interactions.R' 'diagnostics.R' 'metrics.R' 'coefficients.R' 'visualization.R' 'tables.R' 'workflows.R'

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libx11-dev zlib1g-dev

Repository https://ces0491.r-universe.dev

Date/Publication 2026-11-02 16:41:17 UTC

RemoteUrl https://github.com/ces0491/tidylearn

RemoteRef HEAD

RemoteSha 34e62842202bf2863c8dfb172dda4ab50b9a3344

Contents

augment_dbscan 7
augment_hclust 7
augment_kmeans 8
augment_pam 9
augment_pca 9
calc_validation_metrics 10
calc_wss 11
compare_clusterings 11

compare_distances . . . 12
create_cluster_dashboard . . . 13
explore_dbscan_params . . . 14
filter_rules_by_item . . . 14
find_related_items . . . 15
get_pca_loadings . . . 16
get_pca_variance . . . 16
inspect_rules . . . 17
optimal_clusters . . . 18
optimal_hclust_k . . . 18
plot.tidylearn_eda . . . 19
plot.tidylearn_model . . . 20
plot_cluster_comparison . . . 20
plot_cluster_sizes . . . 21
plot_clusters . . . 22
plot_dendrogram . . . 23
plot_distance_heatmap . . . 23
plot_elbow . . . 24
plot_gap_stat . . . 25
plot_knn_dist . . . 25
plot_mds . . . 26
plot_silhouette . . . 27
plot_variance_explained . . . 27
predict.tidylearn_model . . . 28
predict.tidylearn_stratified . . . 29
predict.tidylearn_transfer . . . 30
print.tidy_apriori . . . 30
print.tidy_dbscan . . . 31
print.tidy_gap . . . 32
print.tidy_hclust . . . 32
print.tidy_kmeans . . . 33
print.tidy_mds . . . 34
print.tidy_pam . . . 34
print.tidy_pca . . . 35
print.tidy_silhouette . . . 36
print.tidylearn_automl . . . 36
print.tidylearn_compute_advice . . . 37
print.tidylearn_data . . . 38
print.tidylearn_eda . . . 38
print.tidylearn_gpu_check . . . 39
print.tidylearn_model . . . 40
print.tidylearn_pipeline . . . 40
recommend_products . . . 41
standardize_data . . . 42
suggest_eps . . . 42
summarize_rules . . . 43
summary.tidylearn_model . . . 44
summary.tidylearn_pipeline . . . 45

tidy_apriori . . . 45
tidy_clara . . . 47
tidy_cutree . . . 48
tidy_dbscan . . . 48
tidy_dendrogram . . . 49
tidy_dist . . . 50
tidy_gap_stat . . . 51
tidy_gower . . . 52
tidy_hclust . . . 53
tidy_kmeans . . . 54
tidy_knn_dist . . . 55
tidy_mds . . . 55
tidy_mds_classical . . . 56
tidy_mds_kruskal . . . 57
tidy_mds_sammon . . . 58
tidy_mds_smacof . . . 59
tidy_pam . . . 59
tidy_pca . . . 60
tidy_pca_biplot . . . 61
tidy_pca_screeplot . . . 62
tidy_rules . . . 63
tidy_silhouette . . . 64
tidy_silhouette_analysis . . . 64
tidylearn-classification . . . 65
tidylearn-cloud-consent . . . 65
tidylearn-cloud-cost . . . 66
tidylearn-cloud-endpoint . . . 66
tidylearn-cloud-serialize . . . 66
tidylearn-coefficients . . . 66
tidylearn-core . . . 67
tidylearn-deep-learning . . . 67
tidylearn-diagnostics . . . 67
tidylearn-interactions . . . 67
tidylearn-metrics . . . 68
tidylearn-model-selection . . . 68
tidylearn-neural-networks . . . 68
tidylearn-pipeline . . . 68
tidylearn-read . . . 69
tidylearn-read-backends . . . 70
tidylearn-regression . . . 70
tidylearn-regularization . . . 71
tidylearn-svm . . . 71
tidylearn-tables . . . 71
tidylearn-trees . . . 71
tidylearn-tuning . . . 71
tidylearn-visualization . . . 72
tidylearn-workflows . . . 72
tidylearn-xgboost . . . 72

tl_add_cluster_features . . . 72
tl_anomaly_aware . . . 73
tl_auto_interactions . . . 74
tl_auto_ml . . . 75
tl_calc_classification_metrics . . . 77
tl_check_assumptions . . . 79
tl_check_gpu . . . 79
tl_cloud_allow_host . . . 80
tl_cloud_allowed_hosts . . . 81
tl_cloud_consent . . . 82
tl_cloud_jobs . . . 83
tl_coefficients . . . 83
tl_compare_cv . . . 85
tl_compare_pipeline_models . . . 86
tl_compute_advisor . . . 87
tl_cv . . . 89
tl_dashboard . . . 90
tl_default_param_grid . . . 91
tl_detect_outliers . . . 92
tl_diagnostic_dashboard . . . 93
tl_evaluate . . . 94
tl_explore . . . 95
tl_get_best_model . . . 96
tl_influence_measures . . . 96
tl_interaction_effects . . . 97
tl_load_pipeline . . . 99
tl_model . . . 99
tl_pipeline . . . 102
tl_plot_cv_comparison . . . 103
tl_plot_cv_results . . . 104
tl_plot_deep_architecture . . . 104
tl_plot_deep_history . . . 105
tl_plot_gain . . . 106
tl_plot_importance_comparison . . . 106
tl_plot_importance_regularized . . . 107
tl_plot_influence . . . 108
tl_plot_interaction . . . 109
tl_plot_intervals . . . 110
tl_plot_lift . . . 111
tl_plot_model_comparison . . . 111
tl_plot_nn_architecture . . . 112
tl_plot_nn_tuning . . . 113
tl_plot_partial_dependence . . . 113
tl_plot_regularization_cv . . . 114
tl_plot_regularization_path . . . 115
tl_plot_svm_boundary . . . 115
tl_plot_svm_tuning . . . 116
tl_plot_tree . . . 117

tl_plot_tuning_results . . . 118
tl_plot_xgboost_importance . . . 119
tl_plot_xgboost_shap_dependence . . . 120
tl_plot_xgboost_shap_summary . . . 121
tl_plot_xgboost_tree . . . 121
tl_predict_pipeline . . . 122
tl_prepare_data . . . 123
tl_read . . . 125
tl_read_bigquery . . . 126
tl_read_csv . . . 127
tl_read_db . . . 128
tl_read_dir . . . 128
tl_read_excel . . . 130
tl_read_github . . . 130
tl_read_json . . . 131
tl_read_kaggle . . . 132
tl_read_mysql . . . 133
tl_read_parquet . . . 134
tl_read_postgres . . . 135
tl_read_rdata . . . 136
tl_read_rds . . . 137
tl_read_s3 . . . 138
tl_read_sqlite . . . 139
tl_read_tsv . . . 140
tl_read_zip . . . 140
tl_reduce_dimensions . . . 141
tl_run_pipeline . . . 143
tl_save_pipeline . . . 143
tl_semisupervised . . . 144
tl_split . . . 145
tl_step_selection . . . 146
tl_stratified_models . . . 147
tl_table . . . 149
tl_table_clusters . . . 149
tl_table_coefficients . . . 150
tl_table_comparison . . . 151
tl_table_confusion . . . 152
tl_table_importance . . . 153
tl_table_loadings . . . 153
tl_table_metrics . . . 154
tl_table_variance . . . 155
tl_test_interactions . . . 155
tl_test_model_difference . . . 157
tl_transfer_learning . . . 158
tl_tune_deep . . . 159
tl_tune_grid . . . 160
tl_tune_nn . . . 162
tl_tune_random . . . 163

tl_tune_xgboost 165
tl_version 166
tl_xgboost_shap 167
visualize_rules 167

Index **169**

`augment_dbscan` *Augment Data with DBSCAN Cluster Assignments*

Description

Augment Data with DBSCAN Cluster Assignments

Usage

```
augment_dbscan(dbscan_obj, data)
```

Arguments

`dbscan_obj`	A tidy_dbscan object
`data`	Original data frame

Value

A tibble containing the original `data` with additional columns `cluster` (factor), `is_noise` (logical), and `is_core` (logical).

Examples

```
db <- tidy_dbscan(iris[, 1:4], eps = 0.5, minPts = 5)
augmented <- augment_dbscan(db, iris)
```

`augment_hclust` *Augment Data with Hierarchical Cluster Assignments*

Description

Add cluster assignments to original data

Usage

```
augment_hclust(hclust_obj, data, k = NULL, h = NULL)
```

Arguments

hclust_obj	A tidy_hclust object
data	Original data frame
k	Number of clusters (optional)
h	Height at which to cut (optional)

Value

A tibble containing the original data with an additional cluster integer column indicating cluster assignments.

Examples

```
hc <- tidy_hclust(USArrests, method = "ward.D2")
augmented <- augment_hclust(hc, USArrests, k = 3)
```

---

augment_kmeans *Augment Data with K-Means Cluster Assignments*

Description

Augment Data with K-Means Cluster Assignments

Usage

```
augment_kmeans(kmeans_obj, data)
```

Arguments

kmeans_obj	A tidy_kmeans object
data	Original data frame

Value

A tibble containing the original data with an additional cluster factor column indicating cluster assignments.

Examples

```
km <- tidy_kmeans(iris[, 1:4], k = 3)
augmented <- augment_kmeans(km, iris)
```

augment_pam *Augment Data with PAM Cluster Assignments*

Description

Augment Data with PAM Cluster Assignments

Usage

```
augment_pam(pam_obj, data)
```

Arguments

pam_obj	A tidy_pam object
data	Original data frame

Value

A tibble containing the original `data` with an additional `cluster` factor column indicating cluster assignments.

Examples

```
pm <- tidy_pam(iris[, 1:4], k = 3)
augmented <- augment_pam(pm, iris)
```

augment_pca *Augment Original Data with PCA Scores*

Description

Add PC scores to the original dataset

Usage

```
augment_pca(pca_obj, data, n_components = NULL)
```

Arguments

pca_obj	A tidy_pca object
data	Original data frame
n_components	Number of PCs to add (default: all)

Value

A tibble containing the original `data` with additional columns for each principal component score (named `PC1`, `PC2`, etc.).

Examples

```
pca <- tidy_pca(USArrests)
augmented <- augment_pca(pca, USArrests, n_components = 2)
```

`calc_validation_metrics` *Calculate Cluster Validation Metrics*

Description

Validation metrics for a clustering result

Usage

```
calc_validation_metrics(clusters, data = NULL, dist_mat = NULL)
```

Arguments

`clusters`	Vector of cluster assignments: numeric, factor or character. A label of 0 marks noise, as `tidy_dbscan` reports it: noise points are left out of every measure and counted in `n_noise`.
`data`	Original data frame (for WSS calculation). WSS is taken over its numeric columns, so it needs at least one.
`dist_mat`	Distance matrix (for silhouette)

Value

A single-row tibble with columns `k`, `min_size`, `max_size`, `avg_size`, `n_noise`, and optionally `avg_silhouette`, `min_silhouette` (if `dist_mat` provided; `NA` for a single cluster), and `total_wss` (if `data` provided).

Examples

```
km <- kmeans(iris[, 1:4], centers = 3, nstart = 25)
d <- dist(iris[, 1:4])
metrics <- calc_validation_metrics(km$cluster, iris[, 1:4], d)
```

calc_wss *Calculate Within-Cluster Sum of Squares for Different k*

Description

Used for elbow method to determine optimal k

Usage

```
calc_wss(data, max_k = 10, nstart = 25)
```

Arguments

data	A data frame or tibble
max_k	Maximum number of clusters to test (default: 10)
nstart	Number of random starts for each k (default: 25)

Value

A tibble with columns k (number of clusters) and tot_withinss (total within-cluster sum of squares).

Examples

```
wss <- calc_wss(iris[, 1:4], max_k = 6)
plot(wss$k, wss$tot_withinss, type = "b")
```

compare_clusterings *Compare Multiple Clustering Results*

Description

Compare Multiple Clustering Results

Usage

```
compare_clusterings(cluster_list, data, dist_mat = NULL)
```

Arguments

cluster_list	Named list of cluster assignment vectors. An entry without a name is reported as clustering_<position>.
data	Original data
dist_mat	Distance matrix

Value

A tibble with one row per clustering method and columns for each validation metric (see `calc_validation_metrics`), plus a `method` column identifying the clustering.

Examples

```
km3 <- kmeans(iris[, 1:4], 3, nstart = 25)$cluster
km4 <- kmeans(iris[, 1:4], 4, nstart = 25)$cluster
compare_clusterings(list(k3 = km3, k4 = km4), iris[, 1:4])
```

`compare_distances` *Compare Distance Methods*

Description

Compute distances using multiple methods for comparison

Usage

```
compare_distances(data, methods = c("euclidean", "manhattan", "maximum"))
```

Arguments

`data`	A data frame or tibble
`methods`	Character vector of methods to compare

Value

A named list of `dist` objects, one per method.

Examples

```
dists <- compare_distances(
  iris[, 1:4], methods = c("euclidean", "manhattan")
)
```

`create_cluster_dashboard` *Create Summary Dashboard*

Description

Generate a multi-panel summary of clustering results

Usage

```
create_cluster_dashboard(
  data,
  cluster_col = "cluster",
  validation_metrics = NULL
)
```

Arguments

`data`	Data frame with cluster assignments
`cluster_col`	Cluster column name
`validation_metrics`	Optional tibble of validation metrics

Value

Invisibly returns a named list of the `ggplot` objects drawn: `clusters`, the scatter plot, when the data has two numeric columns besides `cluster_col`; `sizes`; and `metrics`, when `validation_metrics` is given. The combined plot grid is drawn as a side effect via `grid.arrange`.

Examples

```
df <- iris[, 1:4]
df$cluster <- kmeans(df, 3)$cluster
create_cluster_dashboard(df)
```

explore_dbscan_params *Explore DBSCAN Parameters*

Description

Test multiple eps and minPts combinations

Usage

```
explore_dbscan_params(data, eps_values, minPts_values)
```

Arguments

data	A data frame or matrix
eps_values	Vector of eps values to test
minPts_values	Vector of minPts values to test

Value

A tibble with columns eps, minPts, n_clusters, n_noise, and prop_noise for each parameter combination.

Examples

```
params <- explore_dbscan_params(iris[, 1:4],
  eps_values = c(0.3, 0.5, 0.8), minPts_values = c(3, 5))
```

filter_rules_by_item *Filter Rules by Item*

Description

Subset rules containing specific items

Usage

```
filter_rules_by_item(rules_obj, item, where = "both")
```

Arguments

rules_obj	A tidy_apriori object, an arules rules object, or a tibble of rules from tidy_rules, which must keep its lhs_items and rhs_items columns
item	Character; one item name, matched against whole items, so "coffee" does not match "instant coffee"
where	Character; "lhs", "rhs", or "both" (default: "both")

Value

A tibble of rules containing the specified `item` in the requested position.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  filter_rules_by_item(res, "whole milk", where = "rhs")
}
```

`find_related_items` *Find Related Items*

Description

Find items frequently purchased with a given item

Usage

```
find_related_items(rules_obj, item, min_lift = 1.5, top_n = 10)
```

Arguments

`rules_obj`	A tidy_apriori object, an arules rules object, or a tibble of rules from `tidy_rules`, which must keep its `lhs_items` and `rhs_items` columns
`item`	Character; one item name to find associations for, matched against whole items
`min_lift`	Minimum lift threshold (default: 1.5)
`top_n`	Number of top associations to return (default: 10)

Value

A tibble of rules involving the specified `item`, filtered by `min_lift` and sorted by lift in descending order.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  find_related_items(res, "whole milk", min_lift = 1.5)
}
```

get_pca_loadings *Get PCA Loadings in Wide Format*

Description

Get PCA Loadings in Wide Format

Usage

```
get_pca_loadings(pca_obj, n_components = NULL)
```

Arguments

pca_obj	A tidy_pca object from tidy_pca, or a PCA model from tl_model.
n_components	Number of components to include (default: all)

Value

A tibble with one row per variable and one column per principal component, containing the loading values.

Examples

```
pca <- tidy_pca(USArrests)
get_pca_loadings(pca, n_components = 2)
```

get_pca_variance *Get Variance Explained Summary*

Description

Get Variance Explained Summary

Usage

```
get_pca_variance(pca_obj)
```

Arguments

pca_obj	A tidy_pca object from tidy_pca, or a PCA model from tl_model.

Value

A tibble with columns component, sdev, variance, prop_variance, and cum_variance.

Examples

```
pca <- tidy_pca(USArrests)
get_pca_variance(pca)

# The same accessor works on a tl_model() PCA fit
get_pca_variance(tl_model(USArrests, method = "pca"))
```

`inspect_rules` *Inspect Association Rules*

Description

View rules sorted by various quality measures

Usage

```
inspect_rules(rules_obj, by = "lift", n = 10, decreasing = TRUE)
```

Arguments

`rules_obj`	A tidy_apriori object, an arules rules or itemsets object, or a tibble of rules
`by`	Sort by: "support", "confidence", "lift" (default), "count". Itemsets have no lift, so for them the default sorts by support.
`n`	Number of rules to display (default: 10)
`decreasing`	If TRUE (default), the `n` rules with the highest values of `by`, highest first; if FALSE, the `n` with the lowest, lowest first, as in arules' `head(by = )`.

Value

A tibble of the `n` rules ranked highest (or, with `decreasing = FALSE`, lowest) by the quality measure `by`.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  inspect_rules(res, by = "lift", n = 5)
}
```

optimal_clusters *Find Optimal Number of Clusters*

Description

Use multiple methods to suggest optimal k

Usage

```
optimal_clusters(data, max_k = 10, methods = c("silhouette", "gap", "wss"))
```

Arguments

data	A data frame or tibble
max_k	Maximum k to test (default: 10)
methods	Vector of methods: "silhouette", "gap", "wss" (default: all)

Value

A list of class "optimal_k_results" containing one or more of:

- wss: tibble from calc_wss (if "wss" method used)
- silhouette: tibble from tidy_silhouette_analysis (if "silhouette" method used)
- gap: a tidy_gap object from tidy_gap_stat (if "gap" method used)

Examples

```
opt <- optimal_clusters(iris[, 1:4], max_k = 6, methods = "wss")
```

optimal_hclust_k *Determine Optimal Number of Clusters for Hierarchical Clustering*

Description

Use silhouette or gap statistic to find optimal k

Usage

```
optimal_hclust_k(hclust_obj, method = "silhouette", max_k = 10)
```

Arguments

hclust_obj	A tidy_hclust object
method	Character; "silhouette" (default) or "gap". The gap statistic resamples the observations' numeric columns, so it refuses a tree built from a dist object, and one built with distance = "gower" on non-numeric columns.
max_k	Maximum number of clusters to test (default: 10)

Value

A list containing:

- optimal_k: the recommended number of clusters
- method: the evaluation method used
- values: numeric vector of evaluation scores (for silhouette)
- k_range: integer vector of k values tested (for silhouette)

If method = "gap", returns a tidy_gap object instead.

Examples

```
hc <- tidy_hclust(USArrests, method = "ward.D2")
opt <- optimal_hclust_k(hc, method = "silhouette", max_k = 6)
```

---

plot.tidylearn_eda *Plot EDA results*

Description

Plot EDA results

Usage

```
## S3 method for class 'tidylearn_eda'
plot(x, ...)
```

Arguments

x	A tidylearn_eda object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly. Called for its side effect of plotting a PCA scatter plot coloured by cluster.

Examples

```
eda <- tl_explore(iris, response = "Species")
plot(eda)
```

`plot.tidylearn_model` *Plot method for tidylearn models*

Description

Plot method for tidylearn models

Usage

```
## S3 method for class 'tidylearn_model'
plot(x, type = "auto", ...)
```

Arguments

`x`	A tidylearn model object
`type`	Plot type (default: "auto"). An unsupervised model takes only `"auto"`.
`...`	Additional arguments passed to plotting functions

Value

A `ggplot` object. The specific plot depends on the model paradigm and `type` argument.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
plot(model, type = "actual_predicted")
```

`plot_cluster_comparison`
Create Cluster Comparison Plot

Description

Compare multiple clustering results side-by-side

Usage

```
plot_cluster_comparison(data, cluster_cols, x_col, y_col)
```

Arguments

data	Data frame with multiple cluster columns
cluster_cols	Vector of cluster column names
x_col	X-axis variable
y_col	Y-axis variable

Value

The return value of grid.arrange, a gtable drawn as a side effect.

Examples

```
df <- iris[, 1:4]
df$km3 <- kmeans(df, 3)$cluster
df$km4 <- kmeans(df, 4)$cluster
plot_cluster_comparison(df, c("km3", "km4"), "Sepal.Length", "Sepal.Width")
```

---

plot_cluster_sizes *Plot Cluster Size Distribution*

Description

Create bar plot of cluster sizes

Usage

```
plot_cluster_sizes(clusters, title = "Cluster Size Distribution")
```

Arguments

clusters	Vector of cluster assignments
title	Plot title (default: "Cluster Size Distribution")

Value

A ggplot object.

Examples

```
clusters <- kmeans(iris[, 1:4], 3)$cluster
plot_cluster_sizes(clusters)
```

`plot_clusters` *Plot Clusters in 2D Space*

Description

Visualize clustering results using first two dimensions or specified dimensions

Usage

```
plot_clusters(
  data,
  cluster_col = "cluster",
  x_col = NULL,
  y_col = NULL,
  centers = NULL,
  title = "Cluster Plot",
  color_noise_black = TRUE
)
```

Arguments

`data`	A data frame with cluster assignments
`cluster_col`	Name of cluster column (default: "cluster")
`x_col`	X-axis variable (if NULL, uses the first numeric column other than `cluster_col`)
`y_col`	Y-axis variable (if NULL, uses the second numeric column other than `cluster_col`)
`centers`	Optional data frame of cluster centers
`title`	Plot title
`color_noise_black`	If TRUE, color noise points (cluster 0) black

Value

A `ggplot` object.

Examples

```
km <- tidy_kmeans(iris[, 1:4], k = 3)
clustered <- augment_kmeans(km, iris[, 1:4])
plot_clusters(clustered)
```

plot_dendrogram *Plot Dendrogram with Cluster Highlights*

Description

Enhanced dendrogram with colored cluster rectangles

Usage

```
plot_dendrogram(
  hclust_obj,
  k = NULL,
  title = "Hierarchical Clustering Dendrogram"
)
```

Arguments

hclust_obj	Hierarchical clustering object: an hclust, a tidy_hclust, or a tidylearn model fitted with method = "hclust"
k	Number of clusters to highlight
title	Plot title

Value

Invisibly returns the hclust object. The dendrogram is drawn as a side effect.

Examples

```
hc <- hclust(dist(iris[, 1:4]))
plot_dendrogram(hc, k = 3)
```

plot_distance_heatmap *Create Distance Heatmap*

Description

Visualize distance matrix as heatmap

Usage

```
plot_distance_heatmap(
  dist_mat,
  cluster_order = NULL,
  title = "Distance Heatmap"
)
```

Arguments

dist_mat	Distance matrix (dist object)
cluster_order	Optional vector to reorder observations by cluster
title	Plot title

Value

A ggplot object.

Examples

```
d <- dist(iris[1:20, 1:4])
plot_distance_heatmap(d)
```

---

plot_elbow *Create Elbow Plot for K-Means*

Description

Plot total within-cluster sum of squares vs number of clusters

Usage

```
plot_elbow(wss_data, add_line = FALSE, suggested_k = NULL)
```

Arguments

wss_data	A tibble with columns k and tot_withinss (from calc_wss)
add_line	Add vertical line at suggested optimal k? (default: FALSE)
suggested_k	If add_line=TRUE, which k to highlight

Value

A ggplot object.

Examples

```
wss <- data.frame(k = 2:6, tot_withinss = c(150, 90, 60, 50, 45))
plot_elbow(wss)
```

plot_gap_stat *Plot Gap Statistic*

Description

Plot Gap Statistic

Usage

```
plot_gap_stat(gap_obj, show_methods = FALSE)
```

Arguments

gap_obj	A tidy_gap object
show_methods	Logical; show all three k selection methods? (default: FALSE)

Value

A ggplot object.

Examples

```
gap <- tidy_gap_stat(iris[, 1:4], max_k = 6, B = 10)
plot_gap_stat(gap)
```

plot_knn_dist *Plot k-NN Distance Plot*

Description

Visualize k-NN distances to help choose eps

Usage

```
plot_knn_dist(data, k = 4, add_suggestion = TRUE, percentile = 0.95)
```

Arguments

data	A data frame, matrix, or tidy_knn_dist result
k	If data is a data frame, k for k-NN (default: 4)
add_suggestion	Add suggested eps line? (default: TRUE)
percentile	Percentile for suggestion (default: 0.95)

Value

A ggplot object.

Examples

```
plot_knn_dist(iris[, 1:4], k = 5)
```

`plot_mds` *Plot MDS Configuration*

Description

Visualize MDS results

Usage

```
plot_mds(mds_obj, color_by = NULL, label_points = TRUE, dim_x = 1, dim_y = 2)
```

Arguments

`mds_obj`	A tidy_mds object
`color_by`	Optional grouping to colour points by: either a column name present in the MDS configuration, or a vector as long as the data.
`label_points`	Logical; add point labels? (default: TRUE)
`dim_x`	Which dimension for x-axis (default: 1)
`dim_y`	Which dimension for y-axis (default: 2)

Value

A ggplot object.

Examples

```
mds <- tidy_mds(USArrests, method = "classical")
plot_mds(mds)
```

plot_silhouette *Plot Silhouette Analysis*

Description

Plot Silhouette Analysis

Usage

```
plot_silhouette(sil_obj)
```

Arguments

sil_obj	A tidy_silhouette object or tibble from tidy_silhouette_analysis

Value

A ggplot object.

Examples

```
km <- kmeans(iris[, 1:4], centers = 3, nstart = 25)
d <- dist(iris[, 1:4])
sil <- tidy_silhouette(km$cluster, d)
plot_silhouette(sil)
```

plot_variance_explained *Plot Variance Explained (PCA)*

Description

Create combined scree plot showing individual and cumulative variance

Usage

```
plot_variance_explained(variance_tbl, threshold = 0.8)
```

Arguments

variance_tbl	Variance tibble from tidy_pca
threshold	Horizontal line for variance threshold (default: 0.8 for 80%)

Value

A ggplot object.

Examples

```
model <- tl_model(iris[, 1:4], method = "pca")
plot_variance_explained(model$fit$variance_explained)
```

`predict.tidylearn_model`

Predict using a tidylearn model

Description

Unified prediction interface for both supervised and unsupervised models

Usage

```
## S3 method for class 'tidylearn_model'
predict(object, new_data = NULL, type = "response", ...)
```

Arguments

`object`	A tidylearn model object
`new_data`	A data frame containing the new data. If NULL, uses training data.
`type`	Type of prediction, for supervised models only: `"response"` (default), `"prob"` or `"class"`. Note that `"response"` is method-dependent – logistic regression returns probabilities, trees and forests return class labels – so pass `"class"` explicitly when you want labels. `"prob"` and `"class"` need a classification model, and any other value is an error. Ignored by unsupervised models, whose output is determined by the method.
`...`	Additional arguments

Value

For supervised models, a tibble with a `.pred` column; with `type = "prob"`, one column per class instead. It has one row per row of `new_data`, in order, and zero rows for zero-row `new_data`. For unsupervised models, the method's natural output: an `.obs_id` column, the row names of the data predicted on, plus component scores for `"pca"` and `"mds"` (as many as the model keeps), or plus a `cluster` column for the clustering methods.

Unsupervised models differ in whether they can handle new data. `"pca"` projects it and `"kmeans"` assigns it to the nearest centre; `"pam"`, `"clara"`, `"dbscan"`, `"mds"` and `"hclust"` have no out-of-sample projection and error if `new_data` is supplied. For hierarchical clustering, cut the tree with `tidy_cutree()` instead.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
predict(model)
predict(model, new_data = mtcars[1:5, ])
```

`predict.tidylearn_stratified` *Predict from stratified models*

Description

Predict from stratified models

Usage

```
## S3 method for class 'tidylearn_stratified'
predict(object, new_data = NULL, ...)
```

Arguments

`object`	A tidylearn_stratified model object
`new_data`	New data for predictions. NULL predicts the training rows from their stored cluster assignments, which works for every clustering method; new rows can be assigned by k-means only.
`...`	Additional arguments passed to each cluster's model, such as `type`

Value

A tibble of the columns each cluster's model returns for the requested `type` – `.pred` by default, one column per class for `type = "prob"` – and a `.cluster` column with cluster assignments. Rows of a single-class cluster are predicted as that class, with probability 1.

Examples

```
models <- tl_stratified_models(mtcars, mpg ~ .,
  cluster_method = "kmeans", k = 2, supervised_method = "linear")
preds <- predict(models)
```

predict.tidylearn_transfer *Predict with transfer learning model*

Description

Predict with transfer learning model

Usage

```
## S3 method for class 'tidylearn_transfer'
predict(object, new_data, ...)
```

Arguments

object	A tidylearn_transfer model object
new_data	New data for predictions
...	Additional arguments

Value

A tibble with a .pred column containing predictions.

Examples

```
model <- tl_transfer_learning(iris, Species ~ .,
  pretrain_method = "pca", supervised_method = "tree")
preds <- predict(model, iris[1:5, ])
```

print.tidy_apriori *Print Method for tidy_apriori*

Description

Print Method for tidy_apriori

Usage

```
## S3 method for class 'tidy_apriori'
print(x, ...)
```

Arguments

x	A tidy_apriori object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  print(res)
}
```

---

print.tidy_dbscan *Print Method for tidy_dbscan*

Description

Print Method for tidy_dbscan

Usage

```
## S3 method for class 'tidy_dbscan'
print(x, ...)
```

Arguments

x	A tidy_dbscan object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
db <- tidy_dbscan(iris[, 1:4], eps = 0.5, minPts = 5)
print(db)
```

print.tidy_gap *Print Method for tidy_gap*

Description

Print Method for tidy_gap

Usage

```
## S3 method for class 'tidy_gap'
print(x, ...)
```

Arguments

x	A tidy_gap object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
gap <- tidy_gap_stat(iris[, 1:4], max_k = 6, B = 10)
print(gap)
```

print.tidy_hclust *Print Method for tidy_hclust*

Description

Print Method for tidy_hclust

Usage

```
## S3 method for class 'tidy_hclust'
print(x, ...)
```

Arguments

x	A tidy_hclust object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
hc <- tidy_hclust(USArrests, method = "ward.D2")
print(hc)
```

---

print.tidy_kmeans *Print Method for tidy_kmeans*

Description

Print Method for tidy_kmeans

Usage

```
## S3 method for class 'tidy_kmeans'
print(x, ...)
```

Arguments

x	A tidy_kmeans object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
km <- tidy_kmeans(iris[, 1:4], k = 3)
print(km)
```

print.tidy_mds *Print Method for tidy_mds*

Description

Print Method for tidy_mds

Usage

```
## S3 method for class 'tidy_mds'
print(x, ...)
```

Arguments

x	A tidy_mds object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
mds <- tidy_mds(USArrests, method = "classical")
print(mds)
```

print.tidy_pam *Print Method for tidy_pam*

Description

Print Method for tidy_pam

Usage

```
## S3 method for class 'tidy_pam'
print(x, ...)
```

Arguments

x	A tidy_pam object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
pm <- tidy_pam(iris[, 1:4], k = 3)
print(pm)
```

`print.tidy_pca` *Print Method for tidy_pca*

Description

Print Method for tidy_pca

Usage

```
## S3 method for class 'tidy_pca'
print(x, ...)
```

Arguments

x	A tidy_pca object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
pca <- tidy_pca(USArrests)
print(pca)
```

print.tidy_silhouette *Print Method for tidy_silhouette*

Description

Print Method for tidy_silhouette

Usage

```
## S3 method for class 'tidy_silhouette'
print(x, ...)
```

Arguments

x	A tidy_silhouette object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
km <- kmeans(iris[, 1:4], centers = 3, nstart = 25)
d <- dist(iris[, 1:4])
sil <- tidy_silhouette(km$cluster, d)
print(sil)
```

print.tidylearn_automl *Print auto ML results*

Description

Print auto ML results

Usage

```
## S3 method for class 'tidylearn_automl'
print(x, ...)
```

Arguments

x	A tidylearn_automl object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
result <- tl_auto_ml(iris, Species ~ .,
  time_budget = 10,
  use_reduction = FALSE,
  use_clustering = FALSE,
  cv_folds = 2)

# The leaderboard, the winner and the metric it was ranked on
print(result)
```

---

print.tidylearn_compute_advice *Print method for* tidylearn_compute_advice *objects*

Description

Print method for tidylearn_compute_advice objects

Usage

```
## S3 method for class 'tidylearn_compute_advice'
print(x, ...)
```

Arguments

x	A tidylearn_compute_advice object.
...	Unused.

Value

The input x, invisibly.

Examples

```
# Runtime, peak memory and cost per tier, before committing to a fit
advice <- tl_compute_advisor("forest", data = iris, formula = Species ~ .)
print(advice)
```

print.tidylearn_data *Print a tidylearn_data object*

Description

Print a tidylearn_data object

Usage

```
## S3 method for class 'tidylearn_data'
print(x, ...)
```

Arguments

x	A tidylearn_data object.
...	Additional arguments passed to the tibble print method.

Value

The input object x, returned invisibly.

Examples

```
f <- tempfile(fileext = ".csv")
write.csv(iris, f, row.names = FALSE)
d <- tl_read(f)
print(d)
unlink(f)
```

print.tidylearn_eda *Print EDA results*

Description

Print EDA results

Usage

```
## S3 method for class 'tidylearn_eda'
print(x, ...)
```

Arguments

x	A tidylearn_eda object
...	Additional arguments (ignored)

Value

The input object `x`, returned invisibly.

Examples

```
eda <- tl_explore(iris, response = "Species")
print(eda)
```

---

print.tidylearn_gpu_check

Print method for `tidylearn_gpu_check` *objects*

Description

Print method for `tidylearn_gpu_check` objects

Usage

```
## S3 method for class 'tidylearn_gpu_check'
print(x, ...)
```

Arguments

`x`	A `tidylearn_gpu_check` object.
`...`	Unused.

Value

The input `x`, invisibly.

Examples

```
# Reports no GPU rather than failing on a machine without one
print(tl_check_gpu())
```

print.tidylearn_model *Print method for tidylearn models*

Description

Print method for tidylearn models

Usage

```
## S3 method for class 'tidylearn_model'
print(x, ...)
```

Arguments

x	A tidylearn model object
...	Additional arguments (ignored)

Value

The input object x, returned invisibly.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
print(model)
```

print.tidylearn_pipeline *Print a tidylearn pipeline*

Description

Print a tidylearn pipeline

Usage

```
## S3 method for class 'tidylearn_pipeline'
print(x, ...)
```

Arguments

x	A tidylearn pipeline object
...	Additional arguments (not used)

Value

The input pipeline object `x`, returned invisibly.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .)
print(pipe)
```

`recommend_products` *Generate Product Recommendations*

Description

Get product recommendations based on basket contents

Usage

```
recommend_products(rules_obj, basket, top_n = 5, min_confidence = 0.5)
```

Arguments

`rules_obj`	A tidy_apriori object, an arules rules object, or a tibble of rules from `tidy_rules`, which must keep its `lhs_items` and `rhs_items` columns
`basket`	Character vector of items in current basket
`top_n`	Number of recommendations to return (default: 5)
`min_confidence`	Minimum confidence threshold (default: 0.5)

Value

A tibble with columns `rhs` (recommended item), `confidence`, `lift`, and `support`, sorted by lift in descending order. A rule is used when the basket holds its whole left-hand side and none of its right-hand side, and each product is listed once, from its highest-lift rule.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  # The basket has to cover the whole left-hand side of a rule, so a
  # basket of very common items usually matches nothing above the
  # confidence floor
  recommend_products(res, basket = c("flour", "baking powder"))
}
```

standardize_data *Standardize Data*

Description

Center and/or scale numeric variables

Usage

```
standardize_data(data, center = TRUE, scale = TRUE)
```

Arguments

data	A data frame or tibble
center	Logical; center variables? (default: TRUE)
scale	Logical; scale variables to unit variance? (default: TRUE)

Value

A tibble with numeric variables centered and/or scaled as specified; non-numeric columns are returned unchanged. A grouped tibble is standardised within each group, as `dplyr::mutate()` works on it; a rowwise tibble is standardised over its whole columns, since a single value has no spread. Grouping and rowwise identifier columns are left as they are.

Examples

```
std <- standardize_data(iris[, 1:4])
```

suggest_eps *Suggest eps Parameter for DBSCAN*

Description

Use k-NN distance plot to suggest eps value

Usage

```
suggest_eps(data, minPts = 5, method = "percentile", percentile = 0.95)
```

Arguments

data	A data frame or matrix
minPts	The minPts you will pass to tidy_dbscan (default: 5). The k-NN distance is read at k = minPts - 1, the neighbours a core point needs besides itself, as kNNdistplot does.
method	Method to suggest eps: "percentile" (default), "knee"
percentile	If method="percentile", which percentile to use (default: 0.95)

Value

A list containing:

- eps: suggested epsilon value
- knn_distances: full tibble of k-NN distances
- method: method used

Examples

```
eps_info <- suggest_eps(iris, minPts = 5)
eps_info$eps
```

---

summarize_rules *Summarize Association Rules*

Description

Get summary statistics about rules

Usage

```
summarize_rules(rules_obj)
```

Arguments

rules_obj	A tidy_apriori object, an arules rules object, or a rules tibble

Value

A list with n_rules and summary statistics (min, max, mean, median) for support, confidence, and lift.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  summarize_rules(res)
}
```

`summary.tidylearn_model` *Summary method for tidylearn models*

Description

Summary method for tidylearn models

Usage

```
## S3 method for class 'tidylearn_model'
summary(object, ...)
```

Arguments

`object`	A tidylearn model object
`...`	Additional arguments (ignored)

Value

The input `object`, returned invisibly. Called for its side effect of printing model summary and training performance.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
summary(model)
```

summary.tidylearn_pipeline *Summarize a tidylearn pipeline*

Description

Summarize a tidylearn pipeline

Usage

```
## S3 method for class 'tidylearn_pipeline'
summary(object, ...)
```

Arguments

object	A tidylearn pipeline object
...	Additional arguments (not used)

Value

The input pipeline `object`, returned invisibly. Called for its side effect of printing detailed pipeline and model results.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .)
summary(pipe)
```

tidy_apriori *Tidy Apriori Algorithm*

Description

Mine association rules using the Apriori algorithm with tidy output

Usage

```
tidy_apriori(
  transactions,
  support = 0.01,
  confidence = 0.5,
  minlen = 2,
  maxlen = 10,
  target = "rules",
  control = list(verbose = FALSE),
  ...
)
```

Arguments

transactions	A transactions object or data frame
support	Minimum support (default: 0.01)
confidence	Minimum confidence (default: 0.5)
minlen	Minimum rule length (default: 2)
maxlen	Maximum rule length (default: 10)
target	Type of association mined: "rules" (default), "frequent itemsets", "maximally frequent itemsets"
control	A list of algorithmic controls for apriori. The mining trace is off unless the list sets verbose = TRUE; any other entries are passed as given.
...	Further arguments passed to apriori: appearance, to restrict where items may appear, or more mining parameters, such as smax or maxtime, which it adds to the ones above.

Value

A list of class "tidy_apriori" containing:

- rules_tbl: tibble of rules, as tidy_rules returns it, or NULL for an itemset target
- rules: original arules object, rules or itemsets
- parameters: parameters used
- n_rules: number of rules, or of itemsets for an itemset target
- itemsets_tbl: for an itemset target only, a tibble with itemset_id, itemset (its label), size, the quality measures, and items, a list column holding each itemset's items

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
data("Groceries", package = "arules")

# Basic apriori
rules <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)

# Access rules
rules$rules_tbl
}
```

`tidy_clara` *Tidy CLARA (Clustering Large Applications)*

Description

Performs CLARA clustering (scalable version of PAM)

Usage

```
tidy_clara(data, k, metric = "euclidean", samples = 50, sampsize = NULL, ...)
```

Arguments

`data`	A data frame or tibble. CLARA samples observations, so it takes no distance matrix; use `tidy_pam` for a dist object.
`k`	Number of clusters
`metric`	Distance metric (default: "euclidean")
`samples`	Number of samples to draw (default: 50)
`sampsize`	Sample size (default: min(n, 40 + 2*k))
`...`	Further arguments passed to `clara`, such as `correct.d`, `pamLike` or `rngR`. `cluster.only = TRUE` and `medoids.x = FALSE` are refused, under any abbreviation R would accept, since the result needs the fit and its medoids.

Value

A list of class `"tidy_clara"` containing:

- clusters: tibble with observation IDs and cluster assignments
- medoids: tibble of medoid values
- silhouette_avg: average silhouette width
- model: original `clara` object

Examples

```
# CLARA for large datasets
large_data <- iris[rep(1:nrow(iris), 10), 1:4]
clara_result <- tidy_clara(large_data, k = 3, samples = 50)
print(clara_result)
```

tidy_cutree *Cut Hierarchical Clustering Tree*

Description

Cut dendrogram to obtain cluster assignments

Usage

```
tidy_cutree(hclust_obj, k = NULL, h = NULL)
```

Arguments

hclust_obj	A tidy_hclust object or hclust object
k	Number of clusters (optional)
h	Height at which to cut (optional)

Value

A tibble with columns `.obs_id` (observation identifier) and `cluster` (integer cluster assignment).

Examples

```
hc <- tidy_hclust(USArrests, method = "ward.D2")
clusters <- tidy_cutree(hc, k = 3)
```

tidy_dbscan *Tidy DBSCAN Clustering*

Description

Performs density-based clustering with tidy output

Usage

```
tidy_dbscan(data, eps, minPts = 5, cols = NULL, distance = "euclidean")
```

Arguments

data	A data frame, tibble, numeric matrix, or dist object
eps	Neighborhood radius (epsilon)
minPts	Minimum number of points to form a dense region (default: 5)
cols	Columns to include (tidy select). If NULL, uses all numeric columns, or every column for `distance = "gower"`.
distance	Distance metric if data is not a dist object (default: "euclidean"): any method `dist` accepts, or "gower" for mixed data types

Value

A list of class "tidy_dbscan" containing:

- clusters: tibble with observation IDs, cluster assignments (0 = noise), and the logical flags `is_noise` and `is_core`
- summary: tibble with each cluster's size and number of core points
- n_clusters: number of clusters (excluding noise)
- n_noise: number of noise points
- eps, minPts: the parameters used
- model: original dbscan object

Examples

```
# Basic DBSCAN
db_result <- tidy_dbscan(iris, eps = 0.5, minPts = 5)

# With suggested eps from k-NN distance plot
eps_suggestion <- suggest_eps(iris, minPts = 5)
db_result <- tidy_dbscan(iris, eps = eps_suggestion$eps, minPts = 5)
```

`tidy_dendrogram` *Plot Dendrogram*

Description

Create dendrogram visualization

Usage

```
tidy_dendrogram(hclust_obj, k = NULL, hang = 0.01, cex = 0.7)
```

Arguments

`hclust_obj`	A tidy_hclust object or hclust object
`k`	Optional; number of clusters to highlight with rectangles
`hang`	Fraction of plot height to hang labels (default: 0.01)
`cex`	Label size (default: 0.7)

Value

The `hclust` object, returned invisibly. The dendrogram is plotted as a side effect.

Examples

```
hc <- tidy_hclust(USArrests, method = "ward.D2")
tidy_dendrogram(hc, k = 3)
```

`tidy_dist` *Tidy Distance Matrix Computation*

Description

Compute distance matrices with tidy output

Usage

```
tidy_dist(data, method = "euclidean", cols = NULL, ...)
```

Arguments

`data`	A data frame or tibble
`method`	Character; distance method (default: "euclidean"). Options: "euclidean", "manhattan", "maximum", "gower"
`cols`	Columns to include (tidy select). If NULL, uses all numeric columns, or every column for `method = "gower"`.
`...`	Additional arguments passed to distance functions

Value

A `dist` object containing the computed distance matrix.

Examples

```
d <- tidy_dist(iris[, 1:4], method = "euclidean")
```

`tidy_gap_stat` *Tidy Gap Statistic*

Description

Compute gap statistic for determining optimal number of clusters

Usage

```
tidy_gap_stat(data, FUN_cluster = NULL, max_k = 10, B = 50, nstart = 25)
```

Arguments

`data`	A data frame or tibble
`FUN_cluster`	Clustering function (default: uses kmeans internally)
`max_k`	Maximum number of clusters (default: 10)
`B`	Number of bootstrap samples (default: 50)
`nstart`	If using kmeans, number of random starts (default: 25)

Value

A list of class `"tidy_gap"` containing:

- gap_data: tibble with gap statistics for each k
- k_firstSEmax: optimal k via `maxSE`'s firstSEmax method, the smallest k within one standard error of the first local maximum (most conservative)
- k_globalmax: optimal k via the globalmax method, the k with the largest gap (most liberal)
- k_firstmax: optimal k via the firstmax method, the first local maximum of the gap
- recommended_k: recommended k (uses firstSEmax)
- model: the `clusGap` result

Examples

```
gap <- tidy_gap_stat(iris[, 1:4], max_k = 6, B = 10)
gap$recommended_k
```

`tidy_gower` *Gower Distance Calculation*

Description

Computes Gower distance for mixed data types (numeric, factor, ordered)

Usage

```
tidy_gower(data, weights = NULL)
```

Arguments

`data`	A data frame or tibble
`weights`	Optional named vector of variable weights (default: equal weights)

Details

Gower distance handles mixed data types:

- Numeric: range-normalized Manhattan distance
- Factor/Character: 0 if same, 1 if different
- Ordered: treated as numeric ranks

Formula: d_ij = sum(w_k * d_ijk) / sum(w_k) where d_ijk is the dissimilarity for variable k between obs i and j

Value

A `dist` object containing Gower distances, with the `method` attribute set to `"gower"`. A pair of rows with no variable observed in both has no defined distance and is `NA`, as in `daisy`.

Examples

```
# Create example data with mixed types
car_data <- data.frame(
  horsepower = c(130, 250, 180),
  weight = c(1200, 1650, 1420),
  color = factor(c("red", "black", "blue"))
)

# Compute Gower distance
gower_dist <- tidy_gower(car_data)
```

`tidy_hclust` *Tidy Hierarchical Clustering*

Description

Performs hierarchical clustering with tidy output

Usage

```
tidy_hclust(data, method = "average", distance = "euclidean", cols = NULL)
```

Arguments

`data`	A data frame, tibble, or dist object
`method`	Agglomeration method: "ward.D2", "single", "complete", "average" (default), "mcquitty", "median", "centroid"
`distance`	Distance metric if data is not a dist object (default: "euclidean")
`cols`	Columns to include (tidy select). If NULL, uses all numeric columns, or every column for `distance = "gower"`.

Value

A list of class "tidy_hclust" containing:

- model: hclust object
- dist: distance matrix used
- method: linkage method used
- distance_method: distance metric used
- data: original data (for plotting)

Examples

```
# Basic hierarchical clustering
hc_result <- tidy_hclust(USArrests, method = "average")

# With specific distance
hc_result <- tidy_hclust(mtcars, method = "complete", distance = "manhattan")
```

`tidy_kmeans` *Tidy K-Means Clustering*

Description

Performs k-means clustering with tidy output

Usage

```
tidy_kmeans(
  data,
  k,
  cols = NULL,
  nstart = 25,
  iter_max = 100,
  algorithm = "Hartigan-Wong"
)
```

Arguments

`data`	A data frame or tibble
`k`	Number of clusters
`cols`	Columns to include (tidy select). If NULL, uses all numeric columns.
`nstart`	Number of random starts (default: 25)
`iter_max`	Maximum iterations (default: 100)
`algorithm`	K-means algorithm: "Hartigan-Wong" (default), "Lloyd", "Forgy", "MacQueen"

Value

A list of class "tidy_kmeans" containing:

- clusters: tibble with observation IDs and cluster assignments
- centers: tibble of cluster centers
- metrics: tibble with clustering quality metrics
- sizes: integer vector of cluster sizes
- model: original kmeans object

Examples

```
# Basic k-means
km_result <- tidy_kmeans(iris, k = 3)
```

tidy_knn_dist *Compute k-NN Distances*

Description

Calculate distances to k-th nearest neighbor for each point

Usage

```
tidy_knn_dist(data, k = 4, cols = NULL)
```

Arguments

data	A data frame or matrix
k	Number of nearest neighbors (default: 4)
cols	Columns to include (tidy select). If NULL, uses all numeric columns.

Value

A tibble with columns `.obs_id` (observation identifier), `knn_dist` (distance to k-th nearest neighbor), and `rank` (rank of the k-NN distance).

Examples

```
knn <- tidy_knn_dist(iris[, 1:4], k = 5)
```

tidy_mds *Tidy Multidimensional Scaling*

Description

Unified interface for MDS methods with tidy output

Usage

```
tidy_mds(data, method = "classical", ndim = 2, distance = "euclidean", ...)
```

Arguments

data	A data frame, tibble, or distance matrix
method	Character; "classical" (default), "metric", "nonmetric", "sammon", or "kruskal"
ndim	Number of dimensions for output (default: 2)
distance	Character; distance metric if data is not already a dist object (default: "euclidean"): any method dist accepts, on the numeric columns, or "gower" (see tidy_gower), on every column
...	Additional arguments passed to specific MDS functions

Value

A list of class "tidy_mds" containing:

- config: tibble of MDS configuration (coordinates)
- stress: goodness-of-fit measure (if applicable)
- method: character string of method used
- model: original model object

Examples

```
# Classical MDS
mds_result <- tidy_mds(eurodist, method = "classical")
print(mds_result)
```

---

tidy_mds_classical *Classical (Metric) MDS*

Description

Performs classical multidimensional scaling using cmdscale()

Usage

```
tidy_mds_classical(dist_mat, ndim = 2, add_rownames = TRUE)
```

Arguments

dist_mat	A distance matrix (dist object)
ndim	Number of dimensions (default: 2)
add_rownames	Preserve row names from distance matrix (default: TRUE)

Value

A list of class `"tidy_mds"` containing:

- config: tibble of MDS coordinates
- stress: `NA` (not applicable for classical MDS)
- gof: goodness-of-fit (proportion of variance retained)
- eigenvalues: numeric vector of eigenvalues
- method: `"Classical MDS"`
- model: the `cmdscale` result

Examples

```
d <- dist(USArrests)
mds <- tidy_mds_classical(d)
print(mds)
```

---

tidy_mds_kruskal *Kruskal's Non-metric MDS*

Description

Performs Kruskal's isoMDS

Usage

```
tidy_mds_kruskal(dist_mat, ndim = 2, ...)
```

Arguments

`dist_mat`	A distance matrix (dist object)
`ndim`	Number of dimensions (default: 2)
`...`	Additional arguments passed to MASS::isoMDS()

Value

A list of class `"tidy_mds"` containing:

- config: tibble of MDS coordinates
- stress: Kruskal stress value
- method: `"Kruskal's isoMDS"`
- model: the `isoMDS` result

Examples

```
d <- dist(USArrests)
mds <- tidy_mds_kruskal(d)
```

`tidy_mds_sammon` *Sammon Mapping*

Description

Performs Sammon's non-linear mapping

Usage

```
tidy_mds_sammon(dist_mat, ndim = 2, ...)
```

Arguments

`dist_mat`	A distance matrix (dist object)
`ndim`	Number of dimensions (default: 2)
`...`	Additional arguments passed to MASS::sammon()

Value

A list of class `"tidy_mds"` containing:

- config: tibble of MDS coordinates
- stress: Sammon stress value
- method: `"Sammon Mapping"`
- model: the `sammon` result

Examples

```
d <- dist(USArrests)
mds <- tidy_mds_sammon(d)
```

tidy_mds_smacof *SMACOF MDS (Metric or Non-metric)*

Description

Performs MDS using SMACOF algorithm from the smacof package

Usage

```
tidy_mds_smacof(dist_mat, ndim = 2, type = "ratio", ...)
```

Arguments

dist_mat	A distance matrix (dist object)
ndim	Number of dimensions (default: 2)
type	Character; "ratio" for metric, "ordinal" for non-metric (default: "ratio")
...	Additional arguments passed to smacof::mds()

Value

A list of class "tidy_mds" containing:

- config: tibble of MDS coordinates
- stress: stress value from the SMACOF algorithm
- method: character string describing the MDS type
- model: the mds result

Examples

```
d <- dist(USArrests)
mds <- tidy_mds_smacof(d, type = "ratio")
```

tidy_pam *Tidy PAM (Partitioning Around Medoids)*

Description

Performs PAM clustering with tidy output

Usage

```
tidy_pam(data, k, metric = "euclidean", cols = NULL, ...)
```

Arguments

data	A data frame, tibble, or dist object
k	Number of clusters
metric	Distance metric (default: "euclidean"). Use "gower" for mixed data types.
cols	Columns to include (tidy select). If NULL, uses all columns.
...	Further arguments passed to pam, such as nstart, variant or starting medoids. cluster.only = TRUE and diss are refused, under any abbreviation R would accept: the first leaves no fit to build the result from, and tidy_pam() sets the second itself from data.

Value

A list of class "tidy_pam" containing:

- clusters: tibble with observation IDs and cluster assignments
- medoids: tibble with one row per cluster: the medoid's row position in the data (medoid_index, an integer) and, unless data was a dist object, its values
- silhouette_avg: average silhouette width
- silhouette_data: the silhouette information pam() returns (its silinfo)
- model: original pam object

Examples

```
# PAM with Euclidean distance
pam_result <- tidy_pam(iris, k = 3)

# PAM with Gower distance for mixed data
pam_result <- tidy_pam(mtcars, k = 3, metric = "gower")
```

---

tidy_pca *Tidy Principal Component Analysis*

Description

Performs PCA on a dataset using tidyverse principles. Returns a tidy list containing scores, loadings, variance explained, and the original model.

Usage

```
tidy_pca(data, cols = NULL, scale = TRUE, center = TRUE, method = "prcomp")
```

Arguments

data	A data frame or tibble
cols	Columns to include in PCA (tidy select syntax). If NULL, uses all numeric columns.
scale	Logical; should variables be scaled to unit variance? Default TRUE.
center	Logical; should variables be centered? Default TRUE. method = "princomp" always centres, so it warns and records center = TRUE when asked not to.
method	Character; "prcomp" (default, recommended) or "princomp"

Value

A list of class "tidy_pca" containing:

- scores: tibble of PC scores with observation identifiers
- loadings: tibble of variable loadings in long format
- variance: tibble of variance explained by each PC
- model: the original prcomp/princomp object
- settings: list of scale, center, method used

Examples

```
# Basic PCA
pca_result <- tidy_pca(USArrests)


# Access components
pca_result$scores
pca_result$loadings
pca_result$variance
```

---

tidy_pca_biplot *Create PCA Biplot*

Description

Visualize both observations and variables in PC space

Usage

```
tidy_pca_biplot(
  pca_obj,
  pc_x = 1,
  pc_y = 2,
  color_by = NULL,
```

```
  arrow_scale = 1,
  label_obs = FALSE,
  label_vars = TRUE
)
```

Arguments

pca_obj	A tidy_pca object
pc_x	Principal component for x-axis (default: 1)
pc_y	Principal component for y-axis (default: 2)
color_by	Optional grouping to colour points by: either a column name present in the PCA scores, or a vector as long as the data.
arrow_scale	Scaling factor for variable arrows (default: 1)
label_obs	Logical; label observations? (default: FALSE)
label_vars	Logical; label variables? (default: TRUE)

Value

A ggplot object.

Examples

```
pca <- tidy_pca(USArrests)
tidy_pca_biplot(pca)
```

---

tidy_pca_screeplot *Create PCA Scree Plot*

Description

Visualize variance explained by each principal component

Usage

```
tidy_pca_screeplot(pca_obj, type = "proportion", add_line = TRUE)
```

Arguments

pca_obj	A tidy_pca object
type	Character; "variance" or "proportion" (default)
add_line	Logical; add horizontal line at eigenvalue = 1? (for Kaiser criterion)

Value

A ggplot object.

Examples

```
pca <- tidy_pca(USArrests)
tidy_pca_screeplot(pca)
```

`tidy_rules` *Convert Association Rules to Tidy Tibble*

Description

Convert Association Rules to Tidy Tibble

Usage

```
tidy_rules(rules)
```

Arguments

`rules`	A rules object from arules

Value

A tibble with columns `rule_id`, `lhs`, `rhs`, the quality measures (e.g., `support`, `confidence`, `lift`), and the list columns `lhs_items` and `rhs_items`, each rule's items on that side as a character vector. The `lhs` and `rhs` labels are for reading; the helpers that match items read the lists, since an item name can hold the comma that separates items in a label. An empty rule set gives a zero-row tibble with the same columns.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  rules_obj <- arules::apriori(Groceries,
    parameter = list(supp = 0.001, conf = 0.5))
  rules_tbl <- tidy_rules(rules_obj)
}
```

tidy_silhouette *Tidy Silhouette Analysis*

Description

Compute silhouette statistics for cluster validation

Usage

```
tidy_silhouette(clusters, dist_mat)
```

Arguments

clusters	Vector of cluster assignments: numeric, factor or character. Numeric labels, or labels that read as whole numbers (such as the `cluster` factor from `augment_kmeans()`), are kept as they are; other labels are reported as given.
dist_mat	Distance matrix (dist object)

Value

A list of class "tidy_silhouette" containing:

- silhouette_data: tibble with silhouette values for each observation
- avg_width: average silhouette width
- cluster_avg: average silhouette width by cluster

Examples

```
km <- kmeans(iris[, 1:4], centers = 3, nstart = 25)
d <- dist(iris[, 1:4])
sil <- tidy_silhouette(km$cluster, d)
```

tidy_silhouette_analysis *Silhouette Analysis Across Multiple k Values*

Description

Silhouette Analysis Across Multiple k Values

Usage

```
tidy_silhouette_analysis(
  data,
  max_k = 10,
  method = "kmeans",
  nstart = 25,
  dist_method = "euclidean",
  linkage_method = "average"
)
```

Arguments

data	A data frame or tibble
max_k	Maximum number of clusters to test (default: 10)
method	Clustering method: "kmeans" (default) or "hclust"
nstart	If kmeans, number of random starts (default: 25)
dist_method	Distance metric (default: "euclidean")
linkage_method	If hclust, linkage method (default: "average")

Value

A tibble with columns k and avg_sil_width. The "optimal_k" attribute contains the k with the highest average silhouette width.

Examples

```
sil_analysis <- tidy_silhouette_analysis(iris[, 1:4], max_k = 6)
```

---

tidylearn-classification

Classification Functions for tidylearn

Description

Logistic regression and classification metrics functionality

---

tidylearn-cloud-consent

Cloud data-egress consent for tidylearn

Description

The consent gate that stands between a compute = "cloud" call and any training data leaving the user's machine.

`tidylearn-cloud-cost` *Cost controls for tidylearn cloud compute*

Description

Bounding what a cloud fit can cost, and keeping in-flight jobs visible.

`tidylearn-cloud-endpoint`
Cloud endpoint resolution for tidylearn

Description

Resolving and validating the Modal Web Function endpoint that `compute = "cloud"` submits to.

`tidylearn-cloud-serialize`
Model serialisation for tidylearn cloud compute

Description

Converting a fitted tidylearn model to bytes and back, so it can cross the boundary between a remote worker and the caller's R session.

`tidylearn-coefficients`
Coefficient Inference for tidylearn

Description

Coefficients, standard errors and confidence intervals as a tibble, for the methods that have them. `tl_table_coefficients()` formats the same numbers as a `gt` table.

tidylearn-core *tidylearn: A Unified Tidy Interface to R's Machine Learning Ecosystem*

Description

Core functionality for tidylearn. This package provides a unified tidyverse-compatible interface to established R machine learning packages including glmnet, randomForest, xgboost, e1071, rpart, gbm, nnet, cluster, and dbscan. The underlying algorithms are unchanged - tidylearn wraps them with consistent function signatures, tidy tibble output, and unified ggplot2-based visualization. Supervised models keep the wrapped object at model$fit; unsupervised ones put it at modelfitmodel, alongside the tidied components.

tidylearn-deep-learning *Deep Learning for tidylearn*

Description

Deep learning functionality using Keras/TensorFlow

tidylearn-diagnostics *Advanced Diagnostics Functions for tidylearn*

Description

Functions for advanced model diagnostics, assumption checking, and outlier detection

tidylearn-interactions *Interaction Analysis Functions for tidylearn*

Description

Functions for testing, visualizing, and analyzing interactions

`tidylearn-metrics` *Metrics Functionality for tidylearn*

Description

Functions for calculating model evaluation metrics

`tidylearn-model-selection` *Model Selection Functions for tidylearn*

Description

Functions for stepwise model selection, cross-validation, and hyperparameter tuning

`tidylearn-neural-networks` *Neural Networks for tidylearn*

Description

Neural network functionality for classification and regression

`tidylearn-pipeline` *Model Pipeline Functions for tidylearn*

Description

Functions for creating end-to-end model pipelines

`tidylearn-read` *Data Reading Functions for tidylearn*

Description

Functions for reading data from diverse sources into tidy `tidylearn_data` objects. The main dispatcher `tl_read()` auto-detects the format from the file extension and routes to the appropriate reader. All readers return a `tidylearn_data` object, which is a tibble subclass carrying metadata about the data source.

Details

Supported file formats:

- **CSV**: `.csv` files via **readr** (with base R fallback), and `.txt` files named directly
- **TSV**: `.tsv` files via **readr** (with base R fallback)
- **Excel**: `.xls`, `.xlsx`, `.xlsm` files via **readxl**
- **Parquet**: `.parquet` files via **nanoparquet**
- **JSON**: `.json` files, and newline-delimited `.ndjson` files, via **jsonlite**
- **RDS**: `.rds` files via base `readRDS()`
- **RData**: `.rdata`, `.rda` files via base `load()`

CSV and TSV files compressed with gzip, bzip2 or xz (`data.csv.gz`) are recognised by the extension under the compression one.

Supported databases (via **DBI**):

- **SQLite**: `.sqlite`, `.db` files via **RSQLite**
- **PostgreSQL**: via **RPostgres**
- **MySQL/MariaDB**: via **RMariaDB**
- **BigQuery**: `bigquery://project/dataset` URIs via **bigrquery**

Supported cloud/API sources:

- **S3**: `s3://` URIs via **paws.storage**
- **GitHub**: raw file download from repositories
- **Kaggle**: dataset download via Kaggle CLI

A `file://` URL is read as the local path it names. Other web URLs, and URLs with any other scheme such as `ftp://`, are not read; download the file first.

Multi-file reading:

- **Multiple paths**: pass a character vector to `tl_read()`
- **Directories**: `tl_read_dir()` scans for data files with optional pattern/format filtering and recursive scanning

- **Zip archives**: `tl_read_zip()` extracts and reads from `.zip` files

When combining multiple files, a `source_file` column is added to identify the origin of each row: the file's path below the directory or archive it came from, or, for paths given directly, below the deepest folder they share. Files in one folder are labelled by their bare names.

Directory and archive scans read the extensions listed above except `.txt`, which in a folder is as likely to hold notes as data. Name a `.txt` file directly, or select it with `pattern`, to read it.

`tidylearn-read-backends`

Data Reading Backends for tidylearn

Description

Backend readers for databases and cloud/API sources. All backends are optional dependencies checked at call time via `tl_check_packages()`.

Details

Database backends (via **DBI**):

- **SQLite**: via **RSQLite**
- **PostgreSQL**: via **RPostgres**
- **MySQL/MariaDB**: via **RMariaDB**
- **BigQuery**: via **bigrquery**

Cloud/API backends:

- **S3**: via **paws.storage**
- **GitHub**: via base `download.file()`
- **Kaggle**: via Kaggle CLI

`tidylearn-regression` *Regression Functions for tidylearn*

Description

Linear and polynomial regression functionality

tidylearn-regularization *Regularization Functions for tidylearn*

Description

Ridge, Lasso, and Elastic Net regularization functionality

tidylearn-svm *Support Vector Machines for tidylearn*

Description

SVM functionality for classification and regression

tidylearn-tables *Table Functions for tidylearn*

Description

Functions for producing formatted gt tables from tidylearn models. Provides a parallel interface to the plot functions: `tl_table(model, type)` dispatches to the appropriate table formatter based on model type. Requires the gt package (suggested dependency).

tidylearn-trees *Tree-based Methods for tidylearn*

Description

Decision trees, random forests, and boosting functionality

tidylearn-tuning *Hyperparameter Tuning Functions for tidylearn*

Description

Functions for automatic hyperparameter tuning and selection

tidylearn-visualization *Visualization Functions for tidylearn*

Description

General visualization functions for tidylearn models

tidylearn-workflows *High-Level Workflows for Common Machine Learning Patterns*

Description

End-to-end workflows that combine supervised and unsupervised learning.

tidylearn-xgboost *XGBoost Functions for tidylearn*

Description

XGBoost-specific implementation for gradient boosting

tl_add_cluster_features *Cluster-Based Features*

Description

Add cluster assignments as features for supervised learning. This semi-supervised approach can capture non-linear patterns.

Usage

```
tl_add_cluster_features(data, response = NULL, method = "kmeans", ...)
```

Arguments

data	A data frame
response	Response variable name (will be excluded from clustering)
method	Clustering method: "kmeans", "pam", "hclust", "dbscan"
...	Additional arguments for clustering

Value

The original data frame with an additional factor column named `cluster_<method>` containing cluster assignments. The fitted cluster model is stored as an attribute `"cluster_model"`.

Examples

```
# Add cluster features before supervised learning
data_with_clusters <- tl_add_cluster_features(iris, response = "Species",
                                                method = "kmeans", k = 3)
model <- tl_model(data_with_clusters, Species ~ ., method = "forest")
```

---

tl_anomaly_aware *Anomaly-Aware Supervised Learning*

Description

Detect outliers using DBSCAN or other methods, then optionally remove them or down-weight them before supervised learning.

Usage

```
tl_anomaly_aware(
  data,
  formula,
  response,
  anomaly_method = "dbscan",
  action = "flag",
  supervised_method = "tree",
  ...
)
```

Arguments

`data`	A data frame
`formula`	Model formula
`response`	Response variable name, left out of the detection
`anomaly_method`	Method for anomaly detection. Only "dbscan" is implemented; its noise points are the anomalies.
`action`	Action to take: "remove", "flag", "downweight". `"downweight"` gives anomalies a case weight of 0.1, and needs a `supervised_method` that takes case weights: `"linear"`, `"polynomial"`, `"logistic"`, `"tree"`, `"ridge"`, `"lasso"`, `"elastic_net"`, `"forest"`, `"boost"`, `"nn"` or `"xgboost"`. A forest reads them as sampling weights. `"svm"` and `"deep"` take none and are refused.

supervised_method
: Supervised learning method (default: "tree", which handles both regression and classification with any number of classes). "logistic" is binary-only and errors on a response with more than two levels.

...
: Additional arguments for DBSCAN, such as eps and minPts

Details

DBSCAN runs on the predictors the formula names, on their own scale: its eps and minPts (defaults 0.5 and 5, passed through ...) are a distance and a count in those units. If every row comes out as noise the call stops, since no normal data would be left to model.

Value

A tidylearn model object with additional class "tidylearn_anomaly_aware". The model includes an anomaly_info element with anomaly_model, is_anomaly (logical vector), n_anomalies, and action.

Examples

```
model <- tl_anomaly_aware(iris, Species ~ ., response = "Species",
                          anomaly_method = "dbscan", action = "flag")
```

---

tl_auto_interactions *Find important interactions automatically*

Description

Find important interactions automatically

Usage

```
tl_auto_interactions(
  data,
  formula,
  top_n = 3,
  min_r2_change = 0.01,
  max_p_value = 0.05,
  exclude_vars = NULL
)
```

Arguments

data	A data frame containing the data
formula	A formula specifying the base model without interactions
top_n	Number of top interactions to return, a whole number. With 0 none is added.

min_r2_change	Minimum change in R-squared to consider
max_p_value	Maximum p-value for significance
exclude_vars	Character vector of predictor variables that may not appear in a selected interaction. They stay in the model as main effects. Every name must be a predictor in formula.

Value

A tidylearn model object (class "tidylearn_model") fitted with the top significant interaction terms added to the formula. The interaction test results and selected interactions are stored as attributes "interaction_tests" and "selected_interactions", data frames in the layout tl_test_interactions returns. Both are present when no interaction is added, with no rows where there is nothing to report.

Examples

```
model <- tl_auto_interactions(mtcars, mpg ~ wt + hp + cyl, top_n = 2)
```

---

tl_auto_ml *Auto ML: Automated Machine Learning Workflow*

Description

Automatically explores multiple modeling approaches including dimensionality reduction, clustering, and various supervised methods. Returns the best performing model, scored by cross-validation where the time budget allows.

Usage

```
tl_auto_ml(
  data,
  formula,
  task = "auto",
  use_reduction = TRUE,
  use_clustering = TRUE,
  time_budget = 300,
  cv_folds = 5,
  metric = NULL
)
```

Arguments

data	A data frame
formula	Model formula (for supervised learning)

`task` Task type: "classification", "regression", or "auto" (default), which takes it from the response the formula computes, so `factor(am) ~ .` is a classification although `am` is numeric. A factor or character response is a classification and any other a regression. An explicit task has to agree: every candidate but logistic regression takes its task from the response, so a contradicting task would be scored on metrics the candidates cannot produce. A 0/1 numeric response is therefore a regression; convert it with `factor()` to treat its values as classes.

`use_reduction` Whether to try dimensionality reduction (default: TRUE)

`use_clustering` Whether to add cluster features (default: TRUE)

`time_budget` Time budget in seconds (default: 300). The budget is checked between model fits, not during them: once a model starts training it runs to completion, because R cannot safely interrupt C-level code (randomForest, xgboost, e1071). A run can therefore overshoot the budget by the length of the last fit it started.

The budget gates the workflow as follows:

- Baseline models: a tree, with linear regression for a numeric response or logistic regression for a two-class one. A random forest is added when `time_budget` is 30 or more.
- PCA and cluster variants, when enabled: each phase starts only if more than `max(5, 0.1 * time_budget)` seconds remain, and fits one variant per baseline method while at least `max(2, 0.05 * time_budget)` seconds remain.
- Advanced models (SVM and XGBoost for classification, ridge and lasso for regression): only when `time_budget` is 30 or more and more than 40\
- Scoring: a model is cross-validated when more than 30\ budget remains at the moment it is scored, and scored on its own training data otherwise. The leaderboard's `evaluation` column records which.

The example below, with `time_budget = 10` on the three-class `iris`, fits a single tree and cross-validates it.

`cv_folds` Number of cross-validation folds (default: 5). Reducing this (e.g. to 2 or 3) is an effective way to stay closer to the time budget since CV is typically the most expensive step.

`metric` Evaluation metric (default: "accuracy" for classification, "rmse" for regression). Classification takes "accuracy", "precision", "recall", "sensitivity", "specificity", "f1", "auc" or "pr_auc"; regression takes "rmse", "mse", "mae", "mape" or "rsq". It is checked before any model is fitted.

Details

The PCA and cluster variants are built from the formula's predictors only, so a column the formula leaves out (`y ~ . - id`) reaches no candidate. The cluster variants add the cluster assignment to the formula's terms.

Value

A list with class `"tidylearn_automl"` containing:

best_model The best tidylearn model object

models Named list of all successfully trained models

leaderboard Tibble ranking models by the chosen metric, with columns `model`, `score` and `evaluation`. The `evaluation` column records how each score was obtained – `"cv"` for cross-validated, `"train"` for training-set metrics, which are optimistic. Scores of different kinds are not directly comparable; a mixed leaderboard means the budget ran short of cross-validating every model.

task Detected or specified task type

metric Metric used for ranking

runtime Total elapsed time as a difftime object

Examples

```
# Quick run with fast models only (< 30s budget skips forest/SVM/XGBoost)
result <- tl_auto_ml(iris, Species ~ .,
  time_budget = 10,
  use_reduction = FALSE,
  use_clustering = FALSE,
  cv_folds = 2)
result$leaderboard
```

---

tl_calc_classification_metrics

Calculate classification metrics

Description

Scores predicted classes, and optionally class probabilities, against observed classes.

Usage

```
tl_calc_classification_metrics(
  actuals,
  predicted,
  predicted_probs = NULL,
  metrics = c("accuracy", "precision", "recall", "f1", "auc"),
  thresholds = NULL,
  ...
)
```

Arguments

`actuals`	Observed classes: a factor, or a character, logical or numeric vector.
`predicted`	Predicted classes, one for each element of `actuals`.

`predicted_probs`
: Class probabilities, needed for `"auc"`, `"pr_auc"` and `thresholds`: a data frame (or matrix) with a row for each element of `actuals` and a column for each class, named by class – the shape `predict(model, type = "prob")` returns. Without it, `"auc"` and `"pr_auc"` are left out of the result, with a warning when they were asked for by name.

`metrics`
: Character vector of metrics to compute, from `"accuracy"`, `"precision"`, `"recall"`, `"sensitivity"`, `"specificity"`, `"f1"`, `"auc"` and `"pr_auc"`. For more than two classes, `"precision"`, `"recall"`, `"specificity"` and `"f1"` are macro averages, and `"auc"` and `"pr_auc"` average the one-vs-rest areas of the classes.

`thresholds`
: Optional numeric vector of cut-offs on the positive class's probability, for binary classification. Each adds rows scoring the classes that cut-off assigns. Needs `predicted_probs`.

`...`
: Not used.

Details

The classes, in order, are the levels of `predicted` when it is a factor of two or more levels – which is how `predict()` returns them, in the model's order – and otherwise the classes present in `actuals`. Any other class found in `actuals` or `predicted` follows them. For two classes the second is the positive class. `tl_evaluate` instead leaves out rows of a class the model was never trained on, since it knows the model's classes.

A row missing its observed class, its prediction or one of its probabilities is dropped before anything is computed, so every metric describes the same rows. With no row left – `actuals` empty, or every row incomplete – there is nothing to score, and it is an error of class `tidylearn_no_scored_rows`.

Value

A tibble with columns `metric` (character) and `value` (numeric), one row per requested metric. For more than two classes, `"auc"` is followed by an `auc_<class>` row for each class.

`"auc"` and `"pr_auc"` need at least two classes among the scored rows, and are `NA`, with a warning, when there is only one. A class with no scored row has no one-vs-rest area: its `auc_<class>` row is `NA`, the averages cover the other classes, and a warning names it.

With `thresholds`, six rows per cut-off follow – for a cut-off of 0.5, `accuracy_t0.5`, `precision_t0.5`, `recall_t0.5`, `f1_t0.5`, `f2_t0.5` and `f0.5_t0.5` – and the tibble gains a `threshold` column, `NA` on the other rows.

Examples

```
model <- tl_model(iris, Species ~ ., method = "forest")
preds <- predict(model)
tl_calc_classification_metrics(iris$Species, preds$.pred)
```

tl_check_assumptions *Check model assumptions*

Description

Check model assumptions

Usage

```
tl_check_assumptions(model, test = TRUE, verbose = TRUE)
```

Arguments

model	A tidylearn model object
test	Logical; whether to perform statistical tests
verbose	Logical; whether to print test results and explanations

Value

A named list with one element per assumption checked (`linearity`, `independence`, `homoscedasticity`, `normality`, `multicollinearity`, `outliers`), each containing `assumption` (character label), `check` (logical, `NA` when the test could not decide, or `NULL` when no test was run), `details` (character), and `recommendation` (character). An additional `overall` element summarises the number of assumptions checked, violated, and satisfied; an `NA` or `NULL` check counts as neither. A check is `NA` when the fit leaves it nothing to read: on a perfect fit the residuals are rounding error, so every check of them is.

Logistic regression assumes neither normal residuals nor a constant variance, so for a logistic model `normality` and `homoscedasticity` have a `NULL` check and a note saying so. For a factor, multicollinearity is judged on `GVIF^(1/Df)`, the generalised VIF on the scale of an ordinary one.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_check_assumptions(model)
```

tl_check_gpu *Detect local GPU availability for tidylearn methods*

Description

Reports whether the local machine has a CUDA-capable GPU and which tidylearn backends (`xgboost`, `keras3`, `tensorflow`) are positioned to use it. Detection is intentionally cheap: it parses `nvidia-smi` output, giving up with a warning if `nvidia-smi` has not answered within 10 seconds, and reads which R packages are installed from the library without loading them, so it neither starts Python nor fits a model. A backend reported as `gpu_likely_works = TRUE` may still fall back to CPU if it was not compiled or configured with CUDA support — confirm with a small real fit before relying on it for production workloads.

Usage

```
tl_check_gpu(verbose = FALSE)
```

Arguments

`verbose`	Logical. If `TRUE`, also prints the result. Default `FALSE`.

Details

Apple MPS (Metal Performance Shaders) is intentionally not detected in this iteration; see the issue tracker for the MPS feature request.

Value

An object of class `tidylearn_gpu_check`: a list with components `any_gpu` (logical), `cuda` (driver info list with `driver_present`, `device_count`, `device_names`, `driver_version`), `backends` (per-backend status list each containing `installed`, `gpu_likely_works`, `notes`), and `messages` (character vector). A `print()` method is provided.

Examples

```
# Safe to call anywhere: probes for nvidia-smi on the PATH and checks
# which backend packages are installed. Reports no GPU rather than
# failing when there isn't one.
gpu <- tl_check_gpu()
gpu$any_gpu

if (gpu$any_gpu && gpu$backends$xgboost$gpu_likely_works) {
  # xgboost with GPU is worth trying for this workload
}
```

`tl_cloud_allow_host` *Allow an additional host for cloud uploads in this R session*

Description

tidylearn uploads training data only to Modal's own hosts (`*.modal.run`, `*.modal.com`). Modal customers serving Web Functions from a custom domain can add that domain here.

Usage

```
tl_cloud_allow_host(host)
```

Arguments

host	A character vector of host names to allow, or NULL to clear every host added this session. Bare host names of at least three labels only — not URLs, ports, paths or wildcards.

Details

This widens the set of destinations your data may be sent to, so it is deliberately a per-session call rather than an option or an environment variable: a shared .Rprofile or an inherited environment should not be able to add a destination without you writing the call. Additions are never persisted and are forgotten when the session ends.

Hosts match themselves and their subdomains. Adding "fits.example.com" accepts https://fits.example.com and https://a.fits.example.com, and nothing else.

Give the endpoint's full host name. A name of fewer than three labels, such as "example.com" or "co.uk", is refused: two labels can be a public suffix, under which anyone can register a site, and allowing one would admit all of them.

Value

The full allowlist after the change, invisibly.

See Also

tl_cloud_allowed_hosts(), and T9 in system.file("security/threat-model.md", package = "tidylearn").

Examples

```
tl_cloud_allow_host("fits.example.com")
tl_cloud_allowed_hosts()
tl_cloud_allow_host(NULL)
```

---

tl_cloud_allowed_hosts

Hosts tidylearn will currently upload to

Description

Modal's own hosts, plus anything added with tl_cloud_allow_host() during this session.

Usage

```
tl_cloud_allowed_hosts()
```

Value

A character vector of host names.

See Also

`tl_cloud_allow_host()`

Examples

```
tl_cloud_allowed_hosts()
```

`tl_cloud_consent` *Grant or revoke cloud upload consent for this R session*

Description

Fitting with `compute = "cloud"` uploads your training data to your own Modal account, which is a third party. tidylearn will not do that without explicit consent on every call.

Usage

```
tl_cloud_consent(consent = TRUE)
```

Arguments

`consent`	`TRUE` to allow cloud uploads for the rest of the session, `FALSE` to revoke.

Details

There are two ways to give it. Pass `confirm_upload = TRUE` to each `tl_model()` call, or call `tl_cloud_consent()` once to opt in for the rest of the session. The session lock exists for batch and non-interactive work, where a per-call argument is repetitive.

The lock is **not** persisted. It is forgotten when the R session ends, and it is never written to disk. Revoke it early with `tl_cloud_consent(FALSE)`.

tidylearn never prompts interactively for consent, so scripts, CI and `Rscript` behave the same as an interactive session.

Value

The previous consent state, invisibly.

See Also

The full contract is in `system.file("security/threat-model.md", package = "tidylearn")`.

Examples

```
# Opt in for the session, then revoke.
old <- tl_cloud_consent(TRUE)
tl_cloud_consent(FALSE)
```

`tl_cloud_jobs` *Cloud jobs submitted in this R session*

Description

Every cloud submission is recorded here so that no job runs invisibly. A job disappears from this list when its result is collected or it is cancelled.

Usage

```
tl_cloud_jobs()
```

Details

A submitted job runs on Modal whatever this R session does. If the session ends while a job is still listed here, that job keeps running and keeps billing until it finishes or its timeout kills it — this list cannot survive the session, but the timeout does.

Value

A tibble with one row per in-flight job: `call_id`, `method`, `submitted_at`, `timeout_seconds` and `worst_case_cost`. Zero rows when nothing is in flight.

Examples

```
# Nothing in flight in a fresh session
tl_cloud_jobs()
```

`tl_coefficients` *Model coefficients as a tibble*

Description

Returns the coefficients of a fitted tidylearn model as a tibble, with standard errors, test statistics, p-values and – on request – confidence intervals. Available for `"linear"`, `"polynomial"`, `"logistic"`, `"ridge"`, `"lasso"` and `"elastic_net"`. Other methods have no coefficients. `tl_table_importance` covers `"tree"`, `"forest"`, `"boost"` and `"xgboost"`.

Usage

```
tl_coefficients(
  model,
  conf_int = FALSE,
  level = 0.95,
  exponentiate = FALSE,
  lambda = "1se"
)
```

Arguments

model	A tidylearn supervised model object from tl_model.
conf_int	Whether to add conf_low and conf_high columns (default FALSE). Not available for regularised methods.
level	Confidence level for the interval (default 0.95), a number strictly between 0 and 1. Used only when conf_int = TRUE, but checked either way, so a percentage such as 95 is an error.
exponentiate	Whether to report estimate and the interval on the odds scale rather than the log-odds scale (default FALSE). Only meaningful for a classification model, whose coefficients are log odds. The standard error stays on the log-odds scale it was computed on and is renamed std_error_log to say so.
lambda	For regularised methods: "1se" (default), "min", or a numeric penalty value.

Details

Intervals are Wald intervals, computed from the same standard errors as the statistic and p_value columns beside them, so the interval and the p-value always agree about whether zero is excluded. For "linear" and "polynomial" that means t quantiles on the residual degrees of freedom, which is exactly what confint returns for an lm. For "logistic" it means z quantiles, which is what the reported z statistic implies but not what confint() gives – that profiles the likelihood, which is the better interval when the sample is small or a class is nearly separated. Call stats::confint(model$fit) when you want it.

A rank-deficient fit – two perfectly collinear predictors, or an interaction of factors with a combination no row has – cannot estimate every term. Those terms are returned with an NA estimate rather than dropped, so a term named in the formula never disappears from the output without saying so.

Value

A tibble, one row per model term. For "linear", "polynomial" and "logistic": term, estimate, std_error, statistic, p_value, plus conf_low and conf_high when conf_int = TRUE. For regularised methods: term, estimate and the lambda the estimate came from – glmnet reports no standard errors, so there is nothing to test or bound. A multiclass regularised model has one set of coefficients per class, so its rows are led by a class column; exponentiate is not available for it, because glmnet's multinomial coefficients are not relative to a reference class.

See Also

tl_table_coefficients for the same numbers as a formatted table.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_coefficients(model)
tl_coefficients(model, conf_int = TRUE)

# Odds ratios from a logistic fit
am_data <- transform(mtcars, am = factor(am))
model <- tl_model(am_data, am ~ wt, method = "logistic")
tl_coefficients(model, conf_int = TRUE, exponentiate = TRUE)
```

---

tl_compare_cv *Compare models using cross-validation*

Description

Each model is refitted on every fold from its formula, method and fitting arguments. A model a refit would not reproduce is refused: one built by `tl_semisupervised` or `tl_anomaly_aware`, or one of `tl_auto_ml`'s candidates fitted on features it engineered.

Usage

```
tl_compare_cv(data, models, folds = 5, metrics = NULL, ...)
```

Arguments

`data`	A data frame containing the training data
`models`	A named list of supervised tidylearn models, all of them classification or all regression. An unnamed model is named `Model_<position>`.
`folds`	Number of cross-validation folds, a whole number between 2 and `nrow(data)`. `nrow(data)` leaves each row out in turn, and each fold then scores a single prediction. `"accuracy"`, `"mae"`, `"mse"` and `"mape"` average to their values over the left-out predictions. The average `"rmse"` is the mean absolute error; `"precision"`, `"recall"`, `"sensitivity"`, `"specificity"` and `"f1"` are undefined on the folds whose one row gives them nothing to divide by; and `"rsq"`, `"auc"` and `"pr_auc"` are undefined on every fold. A run scoring any of these warns once.
`metrics`	Character vector of metrics to compute, from those `tl_evaluate` computes for the task. Defaults to `c("accuracy", "precision", "recall", "f1", "auc")` for classification and `c("rmse", "mae", "rsq", "mape")` for regression.
`...`	Arguments passed to `tl_model` for every fold fit. Each model is refitted with the arguments it was built with; anything given here overrides them.

Value

A list with two elements:

`$fold_metrics` A data frame with columns `metric`, `value`, `fold`, and `model` containing per-fold results for every model. A metric undefined on a fold – `"auc"` on a fold holding one class – is `NA` there, and so is every metric of a fold none of whose rows can be scored, with a warning naming the fold.

`$summary` A data frame with columns `model`, `metric`, `mean_value`, `sd_value`, `min_value`, and `max_value` summarizing cross-validation performance over the folds with a value. A metric with no value on any fold is `NA` throughout.

Examples

```
m1 <- tl_model(mtcars, mpg ~ wt, method = "linear")
m2 <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
cv <- tl_compare_cv(mtcars, list(simple = m1, full = m2), folds = 3)
cv$summary
```

---

tl_compare_pipeline_models

Compare models from a pipeline

Description

Compare models from a pipeline

Usage

```
tl_compare_pipeline_models(pipeline, metrics = NULL)
```

Arguments

`pipeline`	A tidylearn pipeline object with results
`metrics`	Character vector of metrics to compare, each one the pipeline scored (if NULL, uses all available)

Value

A `ggplot` object showing a faceted bar chart comparing metric values across models, with the best model highlighted.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .,
  models = list(
    tree = list(method = "tree"),
    forest = list(method = "forest", ntree = 100)
  ),
  evaluation = list(validation = "cv", cv_folds = 3))
pipe <- tl_run_pipeline(pipe, verbose = FALSE)

tl_compare_pipeline_models(pipe)

# Restrict the comparison to one metric
tl_compare_pipeline_models(pipe, metrics = "accuracy")
```

---

tl_compute_advisor *Advise on the best compute tier for a tidylearn fit*

Description

Estimates runtime and feasibility on local CPU, local GPU (when available and applicable), and cloud (planning figures only, since cloud submission does not exist yet), then returns a structured recommendation. Useful before kicking off a long fit — call this first to see whether the problem is laptop-sized, GPU-sized, or cloud-sized.

Usage

```
tl_compute_advisor(x, ...)

## S3 method for class 'character'
tl_compute_advisor(
  x,
  data,
  formula = NULL,
  hyperparams = list(),
  gpu_check = NULL,
  ...
)

## S3 method for class 'tidylearn_supervised'
tl_compute_advisor(
  x,
  data = NULL,
  formula = NULL,
  hyperparams = list(),
  gpu_check = NULL,
  ...
```

```
)

## Default S3 method:
tl_compute_advisor(x, ...)
```

Arguments

x	Either a method name (character scalar — same names as accepted by tl_model(), supervised methods only) or a fitted tidylearn_supervised model. When given a fitted model, the advisor introspects its method and formula and advises on what a refit (or a fit on similar new data) would cost.
...	Must be empty: a misspelt argument is an error.
data	A data frame. Required when x is a method name; optional when x is a fitted model (defaults to the model's training data).
formula	Optional formula. Used to determine the number of effective predictors: the terms it expands to against data, so y ~ . - id counts every column but y and id. Ignored when x is a fitted model.
hyperparams	Named list of hyperparameters that affect runtime: degree (polynomial), ntree (forest), n.trees (boost), nrounds (xgboost), size and maxit (nn), and epochs and hidden_layers (deep), e.g. list(nrounds = 1000) or list(epochs = 50, hidden_layers = c(64, 32)). Each must be a positive number, or a vector of them for hidden_layers. Missing entries take the defaults of the method's fit function, so the estimate is for the fit tl_model() would run; other entries are ignored.
gpu_check	Optional tidylearn_gpu_check object. If omitted, tl_check_gpu() is called once internally.

Details

Estimates are deliberately rough. Each method's runtime constant comes from timing fits of three sizes on one laptop, where the estimates land within about a factor of 3 of those fits. Two runs on that laptop differed by up to 2.5x, and other hardware or differently shaped data can be further off, so treat the recommendation as a starting point. The memory multiples and GPU speedups are assumptions with no measurement behind them.

Value

An object of class tidylearn_compute_advice containing problem, local_cpu, local_gpu, cloud, recommendation, and reasoning. A print() method is provided.

Examples

```
# Estimating from a method name needs neither a GPU nor the backend
# package -- it is arithmetic over the problem dimensions
advice <- tl_compute_advisor("xgboost", iris, Species ~ .,
                             hyperparams = list(nrounds = 1000))
advice$recommendation
print(advice)
```

```
# Dispatching on a fitted model requires the backend to be installed.
# CRAN asks examples to use at most two threads.
if (requireNamespace("xgboost", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "xgboost", nthread = 2)
  tl_compute_advisor(model)
}
```

`tl_cv` *Cross-validation for tidylearn models*

Description

Each fold's model is scored with `tl_evaluate`, so the response is read as that function reads it: a transformed left-hand side on its own scale, and classes against the fold model's.

Usage

```
tl_cv(data, formula, method, folds = 5, metrics = NULL, transform = NULL, ...)
```

Arguments

`data`	Data frame
`formula`	Model formula
`method`	Modeling method
`folds`	Number of cross-validation folds, a whole number between 2 and `nrow(data)`. `nrow(data)` leaves each row out in turn, and each fold then scores a single prediction. `"accuracy"`, `"mae"`, `"mse"` and `"mape"` average to their values over the left-out predictions. The average `"rmse"` is the mean absolute error; `"precision"`, `"recall"`, `"sensitivity"`, `"specificity"` and `"f1"` are undefined on the folds whose one row gives them nothing to divide by; and `"rsq"`, `"auc"` and `"pr_auc"` are undefined on every fold. A run scoring any of these warns once.
`metrics`	Character vector of metrics to compute on each fold, passed to `tl_evaluate`. If `NULL` (the default), `tl_evaluate`'s per-task defaults are used.
`transform`	Optional function for feature engineering that has to be refitted per fold. It is called with the training rows of each fold and must return a list with an `apply` function (applied to both the training and assessment rows) and, optionally, a `formula` to fit under. Use this for anything that learns parameters from the data – PCA rotations, cluster centroids, target encodings – since fitting those before the split inflates every fold's score.
`...`	Additional arguments passed to `tl_model` for every fold. Arguments holding one value per row of `data` – `weights`, `subset`, `offset`, `foldid` and `strata` – are refused, since they cannot follow the rows into a fold.

Value

A list with two elements:

`$folds` A list of per-fold evaluation tibbles, each with `metric` and `value` columns.

`$summary` A tibble with columns `metric`, `mean`, and `sd` summarizing performance across folds. A metric undefined on a fold – auc on a fold holding one class – is `NA` there and left out of the mean and sd. So is every metric of a fold none of whose rows can be scored, with a warning giving the reason. A metric with no value on any fold has `NA` mean and sd.

Examples

```
cv <- tl_cv(mtcars, mpg ~ wt + hp, method = "linear", folds = 3)
cv$summary
```

---

tl_dashboard *Create interactive visualization dashboard for a model*

Description

Create interactive visualization dashboard for a model

Usage

```
tl_dashboard(model, new_data = NULL, ...)
```

Arguments

`model`	A tidylearn model object
`new_data`	Optional data frame for evaluation (if NULL, uses training data)
`...`	Additional arguments

Value

A `shinyApp` object.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
app <- tl_dashboard(model)
```

`tl_default_param_grid` *Create pre-defined parameter grids for common models*

Description

Create pre-defined parameter grids for common models

Usage

```
tl_default_param_grid(method, size = "medium", is_classification = TRUE)
```

Arguments

`method`	Model method ("tree", "forest", "boost", "svm", "xgboost", etc.)
`size`	Grid size: "small", "medium", "large"
`is_classification`	Whether the grid is for a classification task (the default) or a regression one. For regression, an `"svm"` grid also tunes `epsilon`, the width of the band within which e1071's regression SVM ignores errors, and the large `"forest"` grid centres `nodesize` on randomForest's regression default of 5 rather than its classification default of 1. The other grids are the same for both tasks.

Value

A named list of parameter values suitable for passing to `tl_tune_grid` or `tl_tune_random`. Each element is a numeric or character vector of candidate values for that hyperparameter, or for `"deep"`'s `hidden_layers` a list of layer-size vectors. The grid is built without the data, so a `"forest"` `mtry` can exceed the number of predictors; the tuners cap it. `"polynomial"` tunes `degree`. The `"xgboost"` grids draw on the values `tl_tune_xgboost` searches by default, and add `nrounds`, which that function chooses by early stopping and `tl_tune_grid` has to tune. `"linear"` and `"logistic"` have no tuneable hyperparameter and return an empty list with a warning, as does an unknown method.

Examples

```
grid <- tl_default_param_grid("tree", size = "small")
grid <- tl_default_param_grid("forest", size = "medium")
grid <- tl_default_param_grid("svm", is_classification = FALSE)
```

`tl_detect_outliers` *Detect outliers in the data*

Description

Detect outliers in the data

Usage

```
tl_detect_outliers(
  data,
  variables = NULL,
  method = "iqr",
  threshold = NULL,
  plot = TRUE
)
```

Arguments

`data`	A data frame containing the data
`variables`	Character vector of variables to check for outliers
`method`	Method for outlier detection: "boxplot", "z-score", "cook", "iqr", "mahalanobis"
`threshold`	Threshold for outlier detection
`plot`	Logical; whether to create a plot of outliers

Value

A list with outlier detection results:

method The detection method used (character).

method_name Human-readable method name (character).

threshold The threshold value used (numeric).

threshold_label Formatted threshold description (character).

outlier_flags A logical matrix (observations x variables), `NA` where a value is missing. For `"cook"` and `"mahalanobis"` a row's flags are the same in every column, and `NA` when any of its values is missing.

any_outlier Logical vector indicating if each observation is an outlier in any variable. Missing flags are ignored, so a row with no flag at all is `FALSE`.

outlier_counts List with `total`, `by_variable`, and `by_observation` counts.

outlier_indices Integer vector of outlier row indices.

plot A `ggplot` object, or `NULL` if `plot = FALSE`.

Examples

```
tl_detect_outliers(mtcars, variables = c("mpg", "wt"), method = "iqr")
```

`tl_diagnostic_dashboard` *Create a diagnostic dashboard*

Description

Create a diagnostic dashboard

Usage

```
tl_diagnostic_dashboard(
  model,
  include_influence = TRUE,
  include_assumptions = TRUE,
  include_performance = TRUE,
  arrange_plots = "grid"
)
```

Arguments

`model`	A tidylearn model object whose fit is an `lm` or `glm`: method `"linear"`, `"polynomial"` or `"logistic"`
`include_influence`	Logical; whether to include influence diagnostics
`include_assumptions`	Logical; whether to include assumption checks
`include_performance`	Logical; whether to include performance metrics
`arrange_plots`	Layout arrangement (e.g., "grid", "row", "column")

Value

A `grid.arrange` object (a `grob`) containing the arranged diagnostic plots.

Examples

```
if (requireNamespace("gridExtra")) {
  model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
  tl_diagnostic_dashboard(model)
}
```

`tl_evaluate` *Evaluate a tidylearn model*

Description

Scores a supervised model's predictions against the observed response.

Usage

```
tl_evaluate(object, new_data = NULL, metrics = NULL, ...)
```

Arguments

`object`	A tidylearn model object
`new_data`	Optional new data for evaluation (if NULL, uses training data)
`metrics`	Character vector of metrics to compute. If `NULL` (the default), `"accuracy"` is used for classification models and `c("rmse", "mae", "rsq")` for regression models. Classification supports `"accuracy"`, `"precision"`, `"recall"`, `"sensitivity"`, `"specificity"`, `"f1"`, `"auc"` and `"pr_auc"`; regression supports `"rmse"`, `"mse"`, `"mae"`, `"mape"` and `"rsq"`. Any other name is an error.
`...`	Additional arguments passed to `predict()`

Details

The observed response is the formula's left-hand side evaluated on the scored rows, so a model of `log(mpg)` is scored against `log(mpg)`, the scale it predicts on. For classification, the observed classes are read against the classes the model was trained on, whose second is the positive class; rows of a class the model never saw are left out with a warning. Rows missing the response or a prediction are dropped. `tl_calc_classification_metrics` describes `"auc"` and `"pr_auc"` when the scored rows hold a single class or lack one.

With no row left to score – `new_data` has no rows, or every row is dropped – it is an error of class `tidylearn_no_scored_rows`, naming the reason. `tl_cv` catches that class and leaves the fold out.

Value

A tibble with columns `metric` (character) and `value` (numeric), containing one row per requested metric, and for `"auc"` on more than two classes an `auc_<class>` row per class as well. An unsupervised model has no response to score against, so it is an error: `calc_validation_metrics` and `tidy_silhouette` assess a clustering.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_evaluate(model)
tl_evaluate(model, metrics = c("rmse", "mape"))
```

`tl_explore` *Exploratory Data Analysis Workflow*

Description

EDA combining unsupervised learning techniques to understand data structure before modeling

Usage

```
tl_explore(data, response = NULL, max_components = 5, k_range = 2:6)
```

Arguments

`data`	A data frame
`response`	Optional response variable for colored visualizations
`max_components`	Maximum number of PCA components to keep (default: 5), or fewer if the data has fewer numeric columns
`k_range`	Range of k values for clustering (default: 2:6). Each is a whole number from 2 to one less than the number of rows, the range a silhouette is defined over.

Value

A list with class `"tidylearn_eda"` containing:

data The original data frame.

response The response variable name, or `NULL`.

pca The fitted PCA model, keeping the first `max_components` components, as `prcomp(rank. = max_components)` would.

optimal_k List with optimal cluster count results.

kmeans The fitted k-means model.

hclust The fitted hierarchical clustering model.

summary List with `n_obs`, `n_vars`, `n_components` (the number kept), and `best_k`.

Examples

```
eda <- tl_explore(iris, response = "Species")
plot(eda)
```

tl_get_best_model *Get the best model from a pipeline*

Description

Get the best model from a pipeline

Usage

```
tl_get_best_model(pipeline)
```

Arguments

pipeline	A tidylearn pipeline object with results

Value

The best `tidylearn_model` object from the pipeline, selected by the metric specified in `evaluation$best_metric`. The model was fitted on the preprocessed training data, so under the default preprocessing it expects predictors imputed and standardised the same way. Predict through `tl_predict_pipeline`, which replays that preprocessing on raw rows; `predict()` on the model itself reads raw values as if they were already standardised, and returns wrong predictions without a warning.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .,
  models = list(tree = list(method = "tree")),
  evaluation = list(metrics = "accuracy", validation = "cv",
    cv_folds = 2, best_metric = "accuracy"))
pipe <- tl_run_pipeline(pipe, verbose = FALSE)
best <- tl_get_best_model(pipe)
best$spec$method

# Predict through the pipeline, which preprocesses the new rows first
tl_predict_pipeline(pipe, iris[c(1, 51, 101), ])
```

tl_influence_measures *Calculate influence measures for a linear model*

Description

Calculate influence measures for a linear model

Usage

```
tl_influence_measures(
  model,
  threshold_cook = NULL,
  threshold_leverage = NULL,
  threshold_dffits = NULL
)
```

Arguments

model	A tidylearn model object
threshold_cook	Cook's distance threshold (default: 4/n)
threshold_leverage	Leverage threshold (default: 2*(p+1)/n)
threshold_dffits	DFFITS threshold (default: 2*sqrt((p+1)/n))

Value

A data frame with one row per observation containing influence measures: cooks_distance, leverage, dffits, std_residual, stud_residual, boolean flags for each threshold (is_cook_influential, is_leverage_influential, is_dffits_influential, is_outlier), per-coefficient dfbetas_* columns, and an overall is_influential flag. Threshold values are stored as attributes.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_influence_measures(model)
```

---

tl_interaction_effects

Calculate partial effects based on a model with interactions

Description

Calculate partial effects based on a model with interactions

Usage

```
tl_interaction_effects(model, var, by_var, at_values = NULL, intervals = TRUE)
```

Arguments

model	A tidylearn model object
var	Variable to calculate effects for
by_var	Variable to calculate effects by (interaction variable)
at_values	Named list of values at which to hold other variables. A variable not named is held at its median if it is numeric, and otherwise at its most frequent value, a tie going to the earlier level.
intervals	Logical; whether to add 95\ lower and upper. This needs a model whose underlying fit is an lm or glm; for any other fit a message is shown and only point estimates are returned. For a glm the interval is built on the link scale and transformed to the response scale.

Value

For numeric var: a list with effects (data frame of predicted values across the variable range for each value of by_var) and slopes (data frame with the slope of var at each value of by_var). For categorical var: a data frame of predicted values at each factor level for each level of by_var. A numeric by_var is evaluated at its quartiles; quartiles that tie are evaluated once, with a label naming each quartile they stand for, such as "Q0/Q25". A factor, character or logical variable is categorical, and is evaluated at the values the data holds: a level no row uses, and a missing value, are left out. A factor is returned as a factor with the model's levels.

fit, lower, upper and slope are on the response scale whatever intervals is set to. For a classification model that is the predicted probability of the second class, so the response must have two classes. slope is the slope of a straight line fitted to fit across the range of var, so for a non-linear link it is an average rate of change over that range. A model variable named fit, by_value or by_label, or with intervals lower or upper, would be overwritten by these columns and is refused.

slopes$slope_se is the standard error of a straight line fitted to the prediction grid, not the sampling uncertainty of the marginal effect. For a linear model the grid is exactly linear in var, so this is near zero by construction and should not be read as a precise estimate. Use summary(model$fit) for inference on the interaction coefficient itself.

Examples

```
model <- tl_model(mtcars, mpg ~ wt * hp, method = "linear")

# How the effect of weight changes across horsepower
effects <- tl_interaction_effects(model, var = "wt", by_var = "hp")
head(effects$effects)
effects$slopes

# slopes$slope_se describes the fitted grid, not the sampling
# uncertainty of the marginal effect -- for that, read the coefficient
summary(model$fit)$coefficients["wt:hp", ]
```

tl_load_pipeline *Load a pipeline from disk*

Description

Load a pipeline from disk

Usage

```
tl_load_pipeline(file)
```

Arguments

file	Path to the pipeline file

Value

A tidylearn_pipeline object previously saved with tl_save_pipeline.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .)
f <- tempfile(fileext = ".rds")
tl_save_pipeline(pipe, f)
pipe2 <- tl_load_pipeline(f)
```

tl_model *Create a tidylearn model*

Description

Unified interface for creating machine learning models by wrapping established R packages. This function dispatches to the appropriate underlying package based on the method.

Usage

```
tl_model(data, formula = NULL, method = "linear", ..., compute = "cpu")
```

Arguments

data	A data frame containing the training data
formula	A formula specifying the model. For unsupervised methods, use ~ vars or NULL; a one-sided formula names columns, and ~ . - x means every numeric column but x. Supervised methods need a two-sided formula.
method	The modeling method. Supervised: "linear" (stats::lm), "polynomial" (stats::lm on polynomial terms), "logistic" (stats::glm), "tree" (rpart), "forest" (randomForest), "boost" (gbm), "ridge"/"lasso"/"elastic_net" (glmnet), "svm" (e1071), "nn" (nnet), "deep" (keras3), "xgboost" (xgboost). The method and the response have to agree, and a mismatch is an error rather than a meaningless fit: "linear" and "polynomial" need a numeric response, "logistic" needs exactly two classes, and every other supervised method takes either. Unsupervised: "pca" (stats::prcomp), "mds" (stats::cmdscale, or smacof or MASS through mds_method), "kmeans" (stats::kmeans), "pam"/"clara" (cluster), "hclust" (stats::hclust), "dbscan" (dbscan).
...	Arguments for the method: see the Method arguments section. Anything else is passed to the underlying model function.
compute	Compute tier for the fit. One of "cpu" (default, existing behaviour), "gpu" (route to local CUDA when the method has an upstream GPU path – xgboost and deep learning today), "auto" (consult tl_compute_advisor and pick per call), or "cloud" (reserved – not yet wired up). When "gpu" is requested for a method without an upstream GPU path or on a machine without a detected GPU, the call falls back to CPU with a warning.

Details

The wrapped packages include: stats (lm, glm, prcomp, kmeans, hclust), glmnet, randomForest, xgboost, gbm, e1071, nnet, rpart, cluster, and dbscan. The underlying algorithms are unchanged - this function provides a consistent interface and returns tidy output.

For a supervised method, model$fit is the object the wrapped function returned. An unsupervised method returns tidied components as well, so the wrapped object sits at modelfitmodel and model$fit is the list holding both.

For classification, the response is reduced to the classes it actually contains: subsetting a data frame keeps every factor level, and a level no row uses would otherwise be reported as a class, given its own (zero) probability column, and counted when deciding whether the problem is binary. The fit is unaffected.

Whether a supervised model is a classification or a regression is decided by the response the formula computes, so factor(cyl) ~ wt is a classification even though cyl is numeric. A factor or text response is a classification. A logical response is a regression for every method but "logistic" – with "linear", a linear probability model – so write factor(y) ~ ... to classify it.

A categorical predictor stored as text, as tl_read() returns it, is made a factor before the fit and stored as one in $data, so every method treats it as a category. At predict(), categorical columns in new data are read against the levels seen in training: new data may hold only some of them, and a level the model was not trained on is an error that names the column.

Value

A tidylearn_model object (S3) containing the fitted model ($fit, or fitmodel for an unsupervised method), model specification ($spec), and training data ($data). update() and step() on $fit refit on the training rows and weights, whatever is in the calling environment. The object also inherits from a method-specific class (e.g., tidylearn_linear) and a paradigm class (tidylearn_supervised or tidylearn_unsupervised).

Method arguments

Arguments in ... are passed to the function the method wraps, except for these, which tidylearn takes itself:

"polynomial" degree (default 2). Each numeric main effect is replaced by poly(term, degree, raw = TRUE) and the result fitted with lm(). A numeric term is one that computes a numeric vector, such as wt or log(wt), or a one-column matrix, such as scale(wt). One that is also part of an interaction keeps its own term and gains I(x^2) up to I(x^degree), so the interaction is coded as written. Factor and other non-numeric terms, interactions, I() terms, bases such as poly() or a spline's, the response as written, an offset() and a removed intercept are kept as they are.

"ridge", "lasso", "elastic_net" alpha, glmnet's mixing parameter (by default 0, 1 and 0.5); lambda, a single penalty to fit at, a sequence of penalties for glmnet::cv.glmnet() to choose from, or NULL (the default) to let it choose its own; and cv_folds (default 5), the number of folds for that cross-validation, which takes the place of glmnet's nfolds. predict() uses the lambda.1se penalty. tidylearn sets x, y, family and nfolds itself and refuses them, along with any argument glmnet does not take.

"svm" tune (default FALSE) and tune_folds (default 5), to choose cost by cross-validation before the fit, with gamma for a non-linear kernel and degree for a polynomial one.

"deep" hidden_layers, activation, dropout, epochs, batch_size, validation_split and learning_rate.

"pca" scale and center (both TRUE).

"mds" mds_method, the variant: "classical" (the default, stats::cmdscale()), "metric" or "nonmetric" (smacof), or "sammon" or "kruskal" (MASS). k, or its alias ndim, is the number of dimensions (default 2).

"kmeans", "pam", "clara" k, the number of clusters (default 3); for "pam", metric as well (default "euclidean").

"hclust" hclust_method, the linkage: "average" (the default), "ward.D", "ward.D2", "single", "complete", "mcquitty", "median" or "centroid"; and distance (default "euclidean").

"dbscan" eps (default 0.5), minPts (default 5) and distance (default "euclidean").

Some defaults differ from the wrapped function's: "forest" computes importance (importance = TRUE), "boost" grows trees of interaction.depth = 3, and "nn" fits size = 5 hidden units with trace = FALSE.

weights and subset take values, one per row of data (such as weights = data$w), not column names. A subset is applied before the fit, and $data holds only the rows it selects. Case weights are applied by every supervised method except "svm" and "deep", which refuse them. An offset, written as offset() in the formula, is applied by "linear", "polynomial" and "logistic"; the other methods refuse one, because their predict() would not add it back.

Examples

```
# Classification -> wraps randomForest::randomForest()
model <- tl_model(iris, Species ~ ., method = "forest")
model$fit  # Access the raw randomForest object

# Regression -> wraps stats::lm()
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
model$fit  # Access the raw lm object

# PCA -> wraps stats::prcomp()
model <- tl_model(iris, ~ ., method = "pca")
model$fit$model  # The raw prcomp object, alongside tidied components

# Clustering -> wraps stats::kmeans()
model <- tl_model(iris, method = "kmeans", k = 3)
model$fit$model  # The raw kmeans object
```

---

tl_pipeline *Create a modeling pipeline*

Description

Create a modeling pipeline

Usage

```
tl_pipeline(
  data,
  formula,
  preprocessing = NULL,
  models = NULL,
  evaluation = NULL,
  ...
)
```

Arguments

data	A data frame containing the data
formula	A formula specifying the model
preprocessing	A named list of preprocessing switches, each TRUE or FALSE: impute_missing (default TRUE) replaces missing predictor values with the training median or mode; standardize (default TRUE) centres and scales numeric predictors where that leaves the model the formula describes unchanged. It leaves alone any column the formula uses inside a function call such as log(), poly() or offset(), every column when the formula has no intercept, and the columns of an interaction whose lower-order terms are not all in the formula; dummy_encode (default TRUE) only records that categorical predictors are encoded by each model's fitting function, and cannot be set to FALSE.

models	A list of models to train
evaluation	A list of evaluation criteria
...	Not used. Anything passed here is an error, so a misspelt argument such as evalution is reported rather than ignored.

Value

A tidylearn_pipeline object (S3 list) with components $formula, $data, $preprocessing, $models, $evaluation, and $results (initially NULL; populated after tl_run_pipeline).

Examples

```
pipe <- tl_pipeline(iris, Species ~ .,
  models = list(tree = list(method = "tree")))
print(pipe)
```

---

tl_plot_cv_comparison *Plot comparison of cross-validation results*

Description

Plot comparison of cross-validation results

Usage

```
tl_plot_cv_comparison(cv_results, metrics = NULL)
```

Arguments

cv_results	Results from tl_compare_cv function
metrics	Character vector of metrics to plot (if NULL, plots all metrics)

Value

A ggplot object showing boxplots of cross-validation metric distributions for each model.

Examples

```
m1 <- tl_model(mtcars, mpg ~ wt, method = "linear")
m2 <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
cv <- tl_compare_cv(mtcars, list(simple = m1, full = m2), folds = 3)
tl_plot_cv_comparison(cv)
```

tl_plot_cv_results *Plot cross-validation results*

Description

Plot cross-validation results

Usage

```
tl_plot_cv_results(cv_results, metrics = NULL)
```

Arguments

cv_results	Cross-validation results from tl_cv function
metrics	Character vector of metrics to plot (if NULL, plots all metrics)

Value

A ggplot object.

Examples

```
cv <- tl_cv(mtcars, mpg ~ wt + hp, method = "linear", folds = 5)
tl_plot_cv_results(cv)

# One metric rather than every one the folds scored
tl_plot_cv_results(cv, metrics = "rmse")
```

tl_plot_deep_architecture *Plot deep learning model architecture*

Description

Plot deep learning model architecture

Usage

```
tl_plot_deep_architecture(model, ...)
```

Arguments

model	A tidylearn deep learning model object
...	Additional arguments passed to the plot() method keras provides for its models, such as to_file or dpi. show_shapes and show_layer_names default to TRUE.

Value

NULL, invisibly. Called for its side effect: keras draws the architecture diagram on the current graphics device, or writes it to to_file. keras renders it through the Python packages pydot and graphviz, and errors saying so when they are not installed.

Examples

```
## Not run:
if (requireNamespace("keras3", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "deep", epochs = 5)
  tl_plot_deep_architecture(model)
}

## End(Not run)
```

---

tl_plot_deep_history *Plot deep learning model training history*

Description

Plot deep learning model training history

Usage

```
tl_plot_deep_history(model, metrics = c("loss", "val_loss"), ...)
```

Arguments

model	A tidylearn deep learning model object
metrics	Which metrics to plot (default: c("loss", "val_loss"))
...	Additional arguments

Value

A ggplot object.

Examples

```
## Not run:
if (requireNamespace("keras3", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "deep", epochs = 5)
  tl_plot_deep_history(model)
}

## End(Not run)
```

tl_plot_gain *Plot gain chart for a classification model*

Description

Plot gain chart for a classification model

Usage

```
tl_plot_gain(model, new_data = NULL, bins = 10, ...)
```

Arguments

model	A tidylearn classification model object
new_data	Optional data frame for evaluation (if NULL, uses training data)
bins	Number of bins for grouping predictions (default: 10)
...	Additional arguments

Value

A ggplot object.

Examples

```
iris_bin <- iris[iris$Species != "setosa", ]
iris_bin$Species <- factor(iris_bin$Species)
model <- tl_model(iris_bin, Species ~ ., method = "logistic")
tl_plot_gain(model)
```

tl_plot_importance_comparison *Plot feature importance across multiple models*

Description

Each model's importance is rescaled on its own: its largest value becomes 100, or, when no value is positive (a forest's permutation importance can be negative throughout), its largest magnitude becomes -100. A factor predictor appears once, under its own name, for every model: the largest of its design columns stands for it where a method ranks those columns separately (ridge, lasso, elastic net and xgboost). A predictor a model was given but did not use scores zero for that model; one it was never given has no bar for it. Tree, forest and boost models are given the variables of an interaction such as wt:hp rather than the interaction itself, so it has no bar for them. Features are ranked on their mean importance over the models that were given them.

Usage

```
tl_plot_importance_comparison(..., top_n = 10, names = NULL)
```

Arguments

...	tidylearn model objects to compare
top_n	Number of top features to display (default: 10)
names	Optional character vector of model names, one unique name per model

Value

A ggplot object.

Examples

```
m1 <- tl_model(iris, Sepal.Length ~ ., method = "forest")
m2 <- tl_model(iris, Sepal.Length ~ ., method = "boost")
tl_plot_importance_comparison(m1, m2, names = c("Forest", "Boost"))
```

---

tl_plot_importance_regularized

Plot variable importance for a regularized model

Description

Plot variable importance for a regularized model

Usage

```
tl_plot_importance_regularized(model, lambda = "1se", top_n = 20, ...)
```

Arguments

model	A tidylearn regularized model object
lambda	Which lambda to use: "1se" (default), "min", or a numeric penalty within the fitted path
top_n	Number of top features to display (default: 20)
...	Additional arguments

Value

A ggplot object.

Examples

```
model <- tl_model(mtcars, mpg ~ ., method = "lasso")
tl_plot_importance_regularized(model)
```

`tl_plot_influence` *Plot influence diagnostics*

Description

Plot influence diagnostics

Usage

```
tl_plot_influence(
  model,
  plot_type = "cook",
  threshold_cook = NULL,
  threshold_leverage = NULL,
  threshold_dffits = NULL,
  n_labels = 3,
  label_size = 3
)
```

Arguments

`model`	A tidylearn model object
`plot_type`	Type of influence plot: "cook", "leverage", "index"
`threshold_cook`	Cook's distance threshold (default: 4/n)
`threshold_leverage`	Leverage threshold (default: 2*(p+1)/n)
`threshold_dffits`	DFFITS threshold (default: 2*sqrt((p+1)/n))
`n_labels`	Number of points to label (default: 3)
`label_size`	Text size for labels (default: 3)

Value

A `ggplot` object.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_plot_influence(model, plot_type = "cook")
```

`tl_plot_interaction` *Plot interaction effects*

Description

Plot interaction effects

Usage

```
tl_plot_interaction(
  model,
  var1,
  var2,
  n_points = 100,
  fixed_values = NULL,
  confidence = TRUE,
  ...
)
```

Arguments

`model`	A tidylearn model object
`var1`	First variable in the interaction
`var2`	Second variable in the interaction
`n_points`	Number of points to use for continuous variables
`fixed_values`	Named list of values for other variables in the model. A variable not named is held at its median if it is numeric, and otherwise at its most frequent value, a tie going to the earlier level.
`confidence`	Logical; whether to show a 95\ band is drawn when one variable is numeric and the other categorical, and needs a model whose underlying fit is an `lm` or `glm`; for a `glm` it is built on the link scale and transformed to the response scale. For any other fit a message says no band was drawn.
`...`	Additional arguments to pass to predict()

Value

A `ggplot` object. Two numeric variables are drawn as a filled contour of the prediction; a numeric and a categorical variable as one line per category; two categorical variables as dodged bars. A factor, character or logical variable is categorical, and only the values the data holds are drawn. For a classification model the prediction is the probability of the second class, so the response must have two classes. The plot's data holds the prediction in a column named `prediction`, and a band in `.lower` and `.upper`, so a model variable with one of those names is refused.

Examples

```
model <- tl_model(mtcars, mpg ~ wt * hp, method = "linear")

# Two numeric variables are drawn as a filled contour over both ranges
tl_plot_interaction(model, var1 = "wt", var2 = "hp")

# A numeric by categorical interaction is drawn as one line per level,
# each with a confidence band
am_model <- tl_model(transform(mtcars, am = factor(am)), mpg ~ wt * am,
  method = "linear")
tl_plot_interaction(am_model, var1 = "wt", var2 = "am")

# Coarser grid, no band
tl_plot_interaction(am_model, var1 = "wt", var2 = "am",
  n_points = 20, confidence = FALSE)
```

---

tl_plot_intervals *Create confidence and prediction interval plots*

Description

Create confidence and prediction interval plots

Usage

```
tl_plot_intervals(model, new_data = NULL, level = 0.95, ...)
```

Arguments

model	A tidylearn regression model object
new_data	Optional data frame for prediction (if NULL, uses training data)
level	Confidence level (default: 0.95)
...	Additional arguments

Value

A ggplot object.

Examples

```
model <- tl_model(mtcars, mpg ~ wt, method = "linear")
tl_plot_intervals(model)
```

tl_plot_lift *Plot lift chart for a classification model*

Description

Plot lift chart for a classification model

Usage

```
tl_plot_lift(model, new_data = NULL, bins = 10, ...)
```

Arguments

model	A tidylearn classification model object
new_data	Optional data frame for evaluation (if NULL, uses training data)
bins	Number of bins for grouping predictions (default: 10)
...	Additional arguments

Value

A ggplot object.

Examples

```
iris_bin <- iris[iris$Species != "setosa", ]
iris_bin$Species <- factor(iris_bin$Species)
model <- tl_model(iris_bin, Species ~ ., method = "logistic")
tl_plot_lift(model)
```

tl_plot_model_comparison *Plot model comparison*

Description

Plot model comparison

Usage

```
tl_plot_model_comparison(..., new_data = NULL, metrics = NULL, names = NULL)
```

Arguments

`...`	tidylearn model objects to compare
`new_data`	Optional data frame for evaluation. If NULL, the models are scored on their training data, which they must share: models fitted on different data are an error asking for `new_data`. A model fitted on engineered features, as `tl_auto_ml()` builds some of its candidates, is scored on the training data of the others.
`metrics`	Character vector of metrics to compute
`names`	Optional character vector of model names

Value

A `ggplot` object.

Examples

```
m1 <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
m2 <- tl_model(mtcars, mpg ~ wt + hp, method = "lasso")
tl_plot_model_comparison(m1, m2, names = c("Linear", "Lasso"))
```

`tl_plot_nn_architecture` *Plot neural network architecture*

Description

Plot neural network architecture

Usage

```
tl_plot_nn_architecture(model, ...)
```

Arguments

`model`	A tidylearn neural network model object
`...`	Additional arguments

Value

The return value of `plotnet`, called for its side effect of drawing the network diagram, or `NULL` if the **NeuralNetTools** package is not installed.

Examples

```
if (requireNamespace("NeuralNetTools", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "nn", size = 3)
  tl_plot_nn_architecture(model)
}
```

tl_plot_nn_tuning *Plot a neural network tuning grid*

Description

Draws the size-by-decay grid as a heatmap of cross-validated error.

Usage

```
tl_plot_nn_tuning(model, ...)
```

Arguments

model	The list returned by tl_tune_nn, not a fitted model — the grid it draws lives in that list's $tuning_results. Anything without that element is refused.
...	Additional arguments

Value

A ggplot object.

Examples

```
tuned <- tl_tune_nn(iris, Species ~ .,
  is_classification = TRUE,
  sizes = c(2, 5), decays = c(0, 0.01), folds = 3)

# The tuning result itself, not tuned$model
tl_plot_nn_tuning(tuned)
```

tl_plot_partial_dependence *Plot partial dependence for tree-based models*

Description

Plot partial dependence for tree-based models

Usage

```
tl_plot_partial_dependence(model, var, n.pts = 20, ...)
```

Arguments

model	A tidylearn tree-based model object
var	Variable name to plot
n.pts	Number of points for continuous variables (default: 20)
...	Additional arguments

Value

A ggplot object. Its data has a var_value column and the mean prediction over the model's training rows, y, at each value. For classification, y is a mean class probability and a class column says which class: the positive class (the second level) alone for a two-class model, and every class, one line each, for more.

Examples

```
model <- tl_model(mtcars, mpg ~ ., method = "forest")
tl_plot_partial_dependence(model, var = "wt")
```

---

tl_plot_regularization_cv

Plot cross-validation results for a regularized model

Description

Shows the cross-validation error as a function of lambda for ridge, lasso, or elastic net models fitted with cv.glmnet.

Usage

```
tl_plot_regularization_cv(model, ...)
```

Arguments

model	A tidylearn regularized model object (ridge, lasso, or elastic_net)
...	Additional arguments (currently unused)

Value

A ggplot object.

Examples

```
model <- tl_model(mtcars, mpg ~ ., method = "ridge")
tl_plot_regularization_cv(model)
```

tl_plot_regularization_path *Plot regularization path for a regularized model*

Description

Plot regularization path for a regularized model

Usage

```
tl_plot_regularization_path(model, label_n = 5, ...)
```

Arguments

model	A tidylearn regularized model object
label_n	Number of top features to label (default: 5)
...	Additional arguments

Value

A ggplot object.

Examples

```
model <- tl_model(mtcars, mpg ~ ., method = "lasso")
tl_plot_regularization_path(model)
```

tl_plot_svm_boundary *Plot SVM decision boundary*

Description

Plot SVM decision boundary

Usage

```
tl_plot_svm_boundary(model, x_var = NULL, y_var = NULL, grid_size = 100, ...)
```

Arguments

model	A tidylearn SVM model object
x_var	Name of the x-axis variable. Defaults to the first numeric predictor in the model's formula.
y_var	Name of the y-axis variable. Defaults to the next numeric predictor in the model's formula.
grid_size	Number of points in each dimension for the grid (default: 100)
...	Additional arguments

Value

A ggplot object. The other predictors are held at their mean, or their most frequent level. A two-class model fitted with probabilities also gets the 0.5 probability contour.

Examples

```
if (requireNamespace("e1071", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "svm")
  tl_plot_svm_boundary(model,
    x_var = "Sepal.Length", y_var = "Sepal.Width")
}
```

---

tl_plot_svm_tuning *Plot SVM tuning results*

Description

Plot SVM tuning results

Usage

```
tl_plot_svm_tuning(model, ...)
```

Arguments

model	A tidylearn SVM model object
...	Additional arguments

Value

A ggplot object.

Examples

```
if (requireNamespace("e1071", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "svm",
    kernel = "linear", tune = TRUE, tune_folds = 2)
  tl_plot_svm_tuning(model)
}
```

`tl_plot_tree` *Plot a decision tree*

Description

Plot a decision tree

Usage

```
tl_plot_tree(model, ...)
```

Arguments

`model`	A tidylearn tree model object
`...`	Additional arguments to pass to rpart.plot()

Value

The return value of `rpart.plot`, called for its side effect of drawing the tree.

Examples

```
model <- tl_model(iris, Species ~ ., method = "tree")
tl_plot_tree(model)
```

`tl_plot_tuning_results` *Plot hyperparameter tuning results*

Description

Plot hyperparameter tuning results

Usage

```
tl_plot_tuning_results(
  model,
  top_n = 5,
  param1 = NULL,
  param2 = NULL,
  plot_type = "scatter"
)
```

Arguments

`model`	A tidylearn model object with tuning results
`top_n`	Number of top parameter sets to highlight
`param1`	First parameter to plot (for 2D grid or scatter plots)
`param2`	Second parameter to plot (for 2D grid or scatter plots)
`plot_type`	Type of plot: "scatter", "grid", "parallel", "importance"

Details

A parameter whose candidates are not single values, such as `hidden_layers = list(c(10), c(20, 10))` or a `parms` list, is drawn as a categorical one, each value labelled as the verbose messages print it. The importance of a numeric parameter is the absolute correlation of its values with the score; that of a categorical one is eta squared from a one-way ANOVA of the score, and 0 when the sets that were scored all share one value.

Value

A `ggplot` object.

Examples

```
model <- tl_tune_grid(iris, Species ~ ., method = "tree",
  param_grid = list(cp = c(0.01, 0.1), minsplit = c(10, 20)),
  folds = 2, verbose = FALSE)
tl_plot_tuning_results(model)
```

`tl_plot_xgboost_importance` *Plot feature importance for an XGBoost model*

Description

Plot feature importance for an XGBoost model

Usage

```
tl_plot_xgboost_importance(model, top_n = 10, importance_type = "gain", ...)
```

Arguments

`model`	A tidylearn XGBoost model object
`top_n`	Number of top features to display (default: 10)
`importance_type`	Type of importance: "gain" (default), "cover", "frequency" or "weight", read from the matching column of `xgboost::xgb.importance()`. A linear booster (`booster = "gblinear"`) reports only "weight", its coefficients, which it uses when `importance_type` is left out; they are ranked by size, and for a multiclass model by their mean size over the classes. A coefficient's size depends on its predictor's scale.
`...`	Additional arguments passed to `xgboost::xgb.importance()`

Value

A `ggplot` object. Its data holds the `top_n` features and their `importance`, relative to the most important feature's.

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  model <- tl_model(mtcars, mpg ~ ., method = "xgboost", nthread = 2)
  tl_plot_xgboost_importance(model)
}
```

`tl_plot_xgboost_shap_dependence` *Plot SHAP dependence for a specific feature*

Description

Plot SHAP dependence for a specific feature

Usage

```
tl_plot_xgboost_shap_dependence(
  model,
  feature,
  interaction_feature = NULL,
  data = NULL,
  n_samples = 100
)
```

Arguments

`model`	A tidylearn XGBoost model object
`feature`	Feature name to plot
`interaction_feature`	Feature to use for coloring (default: NULL)
`data`	Data for SHAP value calculation (default: NULL, uses training data)
`n_samples`	Number of samples to use (default: 100, NULL for all)

Value

A `ggplot` object. A multiclass model is drawn one panel per class, from that class's SHAP values.

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "xgboost", nrounds = 10,
    nthread = 2)

  # One panel per class
  tl_plot_xgboost_shap_dependence(model, feature = "Petal.Length")

  # Colour the points by a second feature to read the interaction
  tl_plot_xgboost_shap_dependence(model,
    feature = "Petal.Length",
    interaction_feature = "Petal.Width")
}
```

tl_plot_xgboost_shap_summary *Plot SHAP summary for XGBoost model*

Description

Plot SHAP summary for XGBoost model

Usage

```
tl_plot_xgboost_shap_summary(model, data = NULL, top_n = 10, n_samples = 100)
```

Arguments

model	A tidylearn XGBoost model object
data	Data for SHAP value calculation (default: NULL, uses training data)
top_n	Number of top features to display (default: 10)
n_samples	Number of samples to use (default: 100, NULL for all)

Value

A ggplot object. Features are ranked by mean absolute SHAP value; a multiclass model is drawn one panel per class.

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  model <- tl_model(mtcars, mpg ~ ., method = "xgboost", nthread = 2)
  tl_plot_xgboost_shap_summary(model, n_samples = 20)
}
```

tl_plot_xgboost_tree *Plot XGBoost tree visualization*

Description

Plot XGBoost tree visualization

Usage

```
tl_plot_xgboost_tree(model, tree_index = 0, ...)
```

Arguments

model	A tidylearn XGBoost model object
tree_index	Index of the tree to plot, counting the first tree as 0 (default: 0). A multiclass model grows one tree per class in each round.
...	Additional arguments passed to xgboost::xgb.plot.tree()

Value

The return value of xgb.plot.tree, a tree diagram rendered via the **DiagrammeR** package.

Examples

```
if (requireNamespace("xgboost", quietly = TRUE) &&
    requireNamespace("DiagrammeR", quietly = TRUE)) {
  model <- tl_model(iris, Species ~ ., method = "xgboost", nrounds = 10,
    nthread = 2)

  # tree_index is zero-based, so this is the first tree
  tl_plot_xgboost_tree(model, tree_index = 0)
}
```

---

tl_predict_pipeline *Make predictions using a pipeline*

Description

Make predictions using a pipeline

Usage

```
tl_predict_pipeline(
  pipeline,
  new_data,
  type = "response",
  model_name = NULL,
  ...
)
```

Arguments

pipeline	A tidylearn pipeline object with results
new_data	A data frame containing the new data
type	Type of prediction (default: "response")
model_name	Name of model to use (if NULL, uses the best model)
...	Additional arguments passed to predict

Value

A tibble with a .pred column containing predictions from the selected (or best) pipeline model, after applying the same preprocessing steps used during training.

Examples

```
train <- iris[c(1:40, 51:90, 101:140), ]
test <- iris[c(41:50, 91:100, 141:150), ]

pipe <- tl_pipeline(train, Species ~ .,
  models = list(
    tree = list(method = "tree"),
    forest = list(method = "forest", ntree = 100)
  ),
  evaluation = list(validation = "cv", cv_folds = 3))
pipe <- tl_run_pipeline(pipe, verbose = FALSE)

# The best model, with the preprocessing learned on the training rows
tl_predict_pipeline(pipe, test)

# Or a named candidate instead of the winner
tl_predict_pipeline(pipe, test, model_name = "tree")
```

---

tl_prepare_data *Data Preprocessing for tidylearn*

Description

Unified preprocessing functions that work with both supervised and unsupervised workflows Prepare Data for Machine Learning

Usage

```
tl_prepare_data(
  data,
  formula = NULL,
  impute_method = "mean",
  scale_method = "standardize",
  encode_categorical = TRUE,
  remove_zero_variance = TRUE,
  remove_correlated = FALSE,
  correlation_cutoff = 0.95
)
```

Arguments

`data`	A data frame. A grouped tibble is prepared as a whole and returned ungrouped.
`formula`	Optional two-sided formula (for supervised learning), whose response must be a column of `data`. Only its predictors are processed; a column it excludes, such as `- id`, is returned unchanged. Leave it `NULL` to process every column.
`impute_method`	Method for imputing a missing numeric value: "mean", "median" or "mode". A missing categorical value is always filled with the column's most frequent value.
`scale_method`	Scaling method: "standardize", "normalize", "robust", "none". A column whose spread is zero or not finite, such as one holding an `Inf`, is left unscaled.
`encode_categorical`	Whether to encode categorical variables (default: TRUE)
`remove_zero_variance`	Remove zero-variance features (default: TRUE)
`remove_correlated`	Remove highly correlated features (default: FALSE)
`correlation_cutoff`	Correlation threshold for removal, greater than 0 and at most 1 (default: 0.95)

Details

Preprocessing pipeline covering imputation, scaling, encoding, and feature engineering

The statistics are learned from, and applied to, the data passed in. Preparing a whole dataset and then splitting it lets the test rows shape the imputation values and scaling their own scores are measured against. To evaluate a model, split first, or use `tl_pipeline`, which learns its preprocessing inside each resampling fold.

Value

A list with components:

`data` The processed data frame.

`original_data` The original unprocessed data frame.

`preprocessing_steps` A record of each step applied (imputation values, encoding maps, scaling parameters, etc.). It is for inspection: no function applies it to new data.

`formula` The formula passed in (or `NULL`).

Examples

```
processed <- tl_prepare_data(iris, Species ~ ., scale_method = "standardize")
model <- tl_model(processed$data, Species ~ ., method = "tree")
```

`tl_read` *Read data from diverse sources*

Description

Auto-detects the data format from the file extension or source pattern and dispatches to the appropriate reader. All readers return a `tidylearn_data` object, which is a tibble subclass carrying metadata about the data source.

Usage

```
tl_read(source, ..., format = NULL, .quiet = FALSE)
```

Arguments

`source`	A file path or `file://` URL, a GitHub or Kaggle URL, an `s3://` or `bigquery://` URI, a database connection string, a directory path, or a character vector of multiple file paths. Other web URLs, and URLs with any other scheme such as `ftp://`, are refused: download the file and read the local copy.
`...`	Additional arguments passed to the format-specific reader.
`format`	Optional explicit format override. One of `"csv"`, `"tsv"`, `"excel"`, `"parquet"`, `"json"`, `"rds"`, `"rdata"`, `"sqlite"`, `"postgres"`, `"mysql"`, `"bigquery"`, `"s3"`, `"github"`, `"kaggle"`. When `NULL` (default), the format is auto-detected from the file extension or source pattern. Note: `.txt` files default to CSV; use `format = "tsv"` to override. A source with a URL scheme other than `file://` is read only with the format its scheme or host implies; to read an S3 object as a given file format, call `tl_read_s3()` with that format.
`.quiet`	Logical. If `TRUE`, suppresses informational messages. Default is `FALSE`.

Details

When `source` is a character vector of multiple paths, each file is read and row-bound into a single result with a `source_file` column giving each file's path below the deepest folder they share. When `source` is a directory path, it is equivalent to calling `tl_read_dir()`. When `source` is a local `.zip` file, it is equivalent to calling `tl_read_zip()`.

Value

A `tidylearn_data` object (a tibble subclass) with attributes `tl_source`, `tl_format`, and `tl_timestamp`.

Examples

```
# The format is detected from the extension
csv <- tempfile(fileext = ".csv")
write.csv(mtcars, csv, row.names = FALSE)
tl_read(csv)
```

```
# Several files are row-bound, with a source_file column naming each
jan <- tempfile(fileext = ".csv")
feb <- tempfile(fileext = ".csv")
write.csv(mtcars[1:16, ], jan, row.names = FALSE)
write.csv(mtcars[17:32, ], feb, row.names = FALSE)
both <- tl_read(c(jan, feb), .quiet = TRUE)
table(both$source_file)

# A .txt file is read as CSV unless told otherwise
txt <- tempfile(fileext = ".txt")
write.table(mtcars, txt, sep = "\t", row.names = FALSE)
tl_read(txt, format = "tsv", .quiet = TRUE)

unlink(c(csv, jan, feb, txt))
```

`tl_read_bigquery` *Read from Google BigQuery*

Description

Executes a SQL query against Google BigQuery and returns the result as a `tidylearn_data` object. Requires the **bigrquery** package and valid Google Cloud authentication.

Usage

```
tl_read_bigquery(project, query, dataset = NULL, ...)
```

Arguments

`project`	Google Cloud project ID, or a `bigquery://project/dataset` URI, whose dataset is used when `dataset` is not given.
`query`	A SQL query string (Standard SQL).
`dataset`	Optional default dataset for unqualified table names, in `project`.
`...`	Additional arguments passed to `bigrquery::bq_project_query()`.

Value

A `tidylearn_data` object containing the query results.

Examples

```
## Not run:
# Needs Google Cloud credentials
tl_read_bigquery(
  project = "my-project",
  query = "SELECT * FROM `my_dataset.my_table` LIMIT 1000"
)
```

```
# Unqualified table names resolve against `dataset`
tl_read_bigquery(
  project = "my-project",
  query = "SELECT * FROM my_table LIMIT 1000",
  dataset = "my_dataset"
)

## End(Not run)
```

`tl_read_csv` *Read a CSV file*

Description

Reads a CSV file into a `tidylearn_data` object. Uses **readr** when available for faster parsing, with a base R fallback.

Usage

```
tl_read_csv(path, ...)
```

Arguments

`path`	Path to a CSV file.
`...`	Additional arguments passed to `readr::read_csv()` or `utils::read.csv()`.

Value

A `tidylearn_data` object (a tibble subclass) with attributes `tl_source`, `tl_format`, and `tl_timestamp`.

Examples

```
path <- tempfile(fileext = ".csv")
write.csv(mtcars, path, row.names = FALSE)
tl_read_csv(path)
unlink(path)
```

tl_read_db *Read from a DBI database connection*

Description

Executes a SQL query against an existing **DBI** connection and returns the result as a `tidylearn_data` object. The connection is not closed by this function — the caller is responsible for managing the connection lifecycle.

Usage

```
tl_read_db(conn, query, ...)
```

Arguments

`conn`	A **DBI** connection object (e.g., from `DBI::dbConnect()`).
`query`	A SQL query string.
`...`	Additional arguments passed to `DBI::dbGetQuery()`.

Value

A `tidylearn_data` object containing the query results.

Examples

```
# RSQLite imports DBI, so both are available here
conn <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
DBI::dbWriteTable(conn, "cars", mtcars)
tl_read_db(conn, "SELECT mpg, cyl, hp FROM cars WHERE cyl = 6")
DBI::dbDisconnect(conn)
```

tl_read_dir *Read all matching files from a directory*

Description

Scans a directory for files matching a pattern or format, reads each one, and row-binds them into a single `tidylearn_data` object with a `source_file` column identifying the origin of each row.

Usage

```
tl_read_dir(
  path,
  pattern = NULL,
  format = NULL,
  recursive = FALSE,
  .quiet = FALSE,
  ...
)
```

Arguments

path	Path to a directory.
pattern	Optional regex pattern to filter file names (e.g., "sales_.*\\.csv$"). If NULL, files are filtered by format instead.
format	File format to read. If NULL and pattern is NULL, all recognized data files are read. If specified, only files with matching extensions are read. .txt files are read only when selected with pattern.
recursive	Logical. Should subdirectories be scanned? Default is FALSE.
.quiet	Suppress messages. Default is FALSE.
...	Additional arguments passed to the format-specific reader.

Value

A tidylearn_data object with an additional source_file column giving each row's file as a path below path, such as "2024/sales.csv".

Examples

```
dir <- tempfile("sales_")
dir.create(file.path(dir, "2024"), recursive = TRUE)
write.csv(mtcars[1:16, ], file.path(dir, "jan.csv"), row.names = FALSE)
write.csv(mtcars[17:32, ], file.path(dir, "2024", "feb.csv"),
          row.names = FALSE)

# Only the top level unless asked to recurse
tl_read_dir(dir, format = "csv")

# Files in subfolders are labelled by their path below dir
all_months <- tl_read_dir(dir, recursive = TRUE, .quiet = TRUE)
table(all_months$source_file)

# Or select files by a regular expression
tl_read_dir(dir, pattern = "^jan", .quiet = TRUE)

unlink(dir, recursive = TRUE)
```

tl_read_excel *Read an Excel file*

Description

Reads an Excel file (`.xls`, `.xlsx`, or `.xlsm`) into a `tidylearn_data` object. Requires the **readxl** package.

Usage

```
tl_read_excel(path, sheet = 1, ...)
```

Arguments

`path`	Path to an Excel file.
`sheet`	Sheet to read. Either a string (the name of a sheet) or an integer (the position of the sheet). Defaults to the first sheet.
`...`	Additional arguments passed to `readxl::read_excel()`.

Value

A `tidylearn_data` object (a tibble subclass) with attributes `tl_source`, `tl_format`, and `tl_timestamp`.

Examples

```
# readxl ships a workbook with one data set per sheet
path <- readxl::readxl_example("datasets.xlsx")
tl_read_excel(path)
tl_read_excel(path, sheet = "mtcars")
```

tl_read_github *Read from GitHub*

Description

Downloads a raw file from a GitHub repository and reads it into a `tidylearn_data` object. Accepts either a full GitHub URL or a `owner/repo` shorthand with a file path.

Usage

```
tl_read_github(source, path = NULL, ref = "main", ..., trust_rds = FALSE)
```

Arguments

source	A GitHub URL or "owner/repo" string. A URL is either a file page (https://github.com/<owner>/<r with raw in place of blob or with neither, and with or without www. and a query such as ?raw=true) or a raw file (https://raw.githubusercontent.com/...), whose query is kept for the download. Zip archives are not read from GitHub: download the file and read it with tl_read_zip().
path	Path to the file within the repository (required when source is "owner/repo" format).
ref	Branch, tag, or commit SHA. Default is "main".
...	Additional arguments passed to the format-specific reader.
trust_rds	Logical. Read an .rds, .rdata or .rda file on R older than 4.4.0? Those versions can run code embedded in a crafted file as it is read (CVE-2024-27322), so such files are refused there unless this is TRUE. It has no effect on R 4.4.0 or later. Default FALSE.

Value

A tidylearn_data object containing the downloaded data.

Reading R serialisation files

An .rds or .rdata file is rebuilt with readRDS() or load(), which recreate whatever R objects the file describes. Read them only from a repository you trust, on any version of R.

Examples

```
## Not run:
# Downloads over the network
tl_read_github("user/repo", path = "data/file.csv")
tl_read_github("https://github.com/user/repo/blob/main/data/file.csv")

## End(Not run)
```

---

tl_read_json *Read a JSON file*

Description

Reads a JSON file into a tidylearn_data object. Expects the JSON to represent tabular data (array of objects or similar). A file with the .ndjson extension is read as newline-delimited JSON, one record per line, and so is a .json file that does not parse as a single document but does as one record per line. Requires the **jsonlite** package.

Usage

```
tl_read_json(path, flatten = TRUE, ...)
```

Arguments

path	Path to a JSON file.
flatten	Logical. Automatically flatten nested data frames? Default is TRUE.
...	Additional arguments passed to jsonlite::fromJSON(), or to jsonlite::stream_in() for an .ndjson file.

Value

A tidylearn_data object (a tibble subclass) with attributes tl_source, tl_format, and tl_timestamp.

Examples

```
path <- tempfile(fileext = ".json")
jsonlite::write_json(mtcars, path)
tl_read_json(path)

# Newline-delimited JSON: one record per line
lines <- tempfile(fileext = ".ndjson")
jsonlite::stream_out(mtcars, file(lines), verbose = FALSE)
tl_read_json(lines)

unlink(c(path, lines))
```

---

tl_read_kaggle *Read from Kaggle*

Description

Downloads a dataset file from Kaggle using the Kaggle CLI and reads it into a tidylearn_data object. Requires the Kaggle CLI to be installed and configured (pip install kaggle).

Usage

```
tl_read_kaggle(source, file = NULL, dest = NULL, type = "dataset", ...)
```

Arguments

source	A Kaggle dataset slug (e.g., "user/dataset-name") or a Kaggle URL. A URL may point at any tab of the dataset or competition page (/data, /code, /versions/2) and may carry a query; a competition URL is read as a competition without type being set.
file	The specific file to read from the dataset, as a path within it; a file the CLI saved under its base name is found too. If NULL, the download is searched for files these readers handle (CSV, TSV, Excel, Parquet and JSON, with compressed CSV/TSV and .ndjson); with several, the newest is read and a message names it.

dest	Directory to keep the download in. The default is a fresh per-dataset directory under tempdir(). A supplied dest receives this download's files, replacing files of the same name; nothing else in it is read, unpacked or changed.
type	Either "dataset" (default) or "competition". Left unset, a competition URL in source sets it.
...	Additional arguments passed to the format-specific reader.

Value

A tidylearn_data object containing the downloaded data.

Examples

```
## Not run:
# Needs the Kaggle CLI and Kaggle credentials
tl_read_kaggle("zillow/zecon", file = "Zip_time_series.csv")
tl_read_kaggle("titanic", file = "train.csv", type = "competition")
tl_read_kaggle("https://www.kaggle.com/competitions/titanic/data",
               file = "train.csv")

## End(Not run)
```

---

tl_read_mysql *Read from a MySQL/MariaDB database*

Description

Connects to a MySQL or MariaDB database, executes a SQL query, and returns the result as a tidylearn_data object. Accepts either a connection string or individual connection parameters. Requires **DBI** and **RMariaDB**.

Usage

```
tl_read_mysql(
  dsn,
  query,
  dbname = NULL,
  user = NULL,
  password = NULL,
  port = 3306,
  ...
)
```

Arguments

dsn	A MySQL connection string (e.g., "mysql://user:pass@host:port/dbname"), or the database host if using named parameters. Percent-encode special characters in the user name and password (@ as %40). Query parameters are refused, because **RMariaDB** ignores arguments it does not know; pass its dbConnect() arguments, such as ssl.ca, through ....
query	A SQL query string.
dbname	Database name (if not in dsn).
user	Username (if not in dsn).
password	Password (if not in dsn), which keeps it out of the connection string.
port	Port number (if not in dsn). Default is 3306.
...	Additional arguments passed to DBI::dbConnect().

Value

A tidylearn_data object containing the query results.

Examples

```
## Not run:
# Needs a running MySQL or MariaDB server
tl_read_mysql(
  dsn = "localhost",
  query = "SELECT * FROM my_table",
  dbname = "mydb",
  user = "myuser",
  password = Sys.getenv("MYSQL_PWD")
)

## End(Not run)
```

---

tl_read_parquet *Read a Parquet file*

Description

Reads a Parquet file into a tidylearn_data object. Requires the **nanoparquet** package.

Usage

```
tl_read_parquet(path, ...)
```

Arguments

path	Path to a Parquet file.
...	Additional arguments passed to nanoparquet::read_parquet().

Value

A `tidylearn_data` object (a tibble subclass) with attributes `tl_source`, `tl_format`, and `tl_timestamp`.

Examples

```
path <- tempfile(fileext = ".parquet")
nanoparquet::write_parquet(mtcars, path)
tl_read_parquet(path)
unlink(path)
```

---

tl_read_postgres *Read from a PostgreSQL database*

Description

Connects to a PostgreSQL database, executes a SQL query, and returns the result as a `tidylearn_data` object. Accepts either a connection string or individual connection parameters. Requires **DBI** and **RPostgres**.

Usage

```
tl_read_postgres(
  dsn,
  query,
  dbname = NULL,
  user = NULL,
  password = NULL,
  port = 5432,
  ...
)
```

Arguments

`dsn`	A PostgreSQL connection string (e.g., `"postgres://user:pass@host:port/dbname"`), or the database host if using named parameters. Percent-encode special characters in the user name and password (`@` as `%40`). Query parameters such as `?sslmode=require` are passed to libpq as connection keywords.
`query`	A SQL query string.
`dbname`	Database name (if not in `dsn`).
`user`	Username (if not in `dsn`).
`password`	Password (if not in `dsn`), which keeps it out of the connection string.
`port`	Port number. Default is 5432.
`...`	Additional arguments passed to `DBI::dbConnect()`.

Value

A `tidylearn_data` object containing the query results.

Examples

```
## Not run:
# Needs a running PostgreSQL server
tl_read_postgres(
  dsn = "localhost",
  query = "SELECT * FROM my_table",
  dbname = "mydb",
  user = "myuser",
  password = Sys.getenv("PGPASSWORD")
)

# The same connection as a URL, with the password still kept out of it
tl_read_postgres(
  "postgres://myuser@localhost:5432/mydb?sslmode=require",
  query = "SELECT * FROM my_table",
  password = Sys.getenv("PGPASSWORD")
)

## End(Not run)
```

`tl_read_rdata` *Read an RData file*

Description

Reads an RData (`.rdata` or `.rda`) file into a `tidylearn_data` object. Since RData files can contain multiple objects, use the `name` argument to specify which object to extract. If `name` is `NULL` and the file contains exactly one data frame, it is returned automatically.

Usage

```
tl_read_rdata(path, name = NULL, ...)
```

Arguments

`path`	Path to an RData file.
`name`	Optional name of the object to extract from the RData file. If `NULL` (default), the function returns the first data frame found, or errors if there are multiple data frames.
`...`	Currently unused.

Value

A `tidylearn_data` object (a tibble subclass) with attributes `tl_source`, `tl_format`, and `tl_timestamp`.

Reading files you did not create

load() rebuilds whatever R objects the file describes, so read only files from a source you trust. On R before 4.4.0 a crafted file can run code as it is read (CVE-2024-27322). The remote readers tl_read_github() and tl_read_s3() refuse .rds and .rdata files on those versions unless trust_rds = TRUE.

Examples

```
path <- tempfile(fileext = ".rdata")
cars <- mtcars
flowers <- iris
save(cars, flowers, file = path)

# With more than one data frame in the file, name the one to read
tl_read_rdata(path, name = "flowers")
unlink(path)
```

---

tl_read_rds *Read an RDS file*

Description

Reads an RDS file into a tidylearn_data object. Uses base R readRDS() — no additional packages required.

Usage

```
tl_read_rds(path)
```

Arguments

path	Path to an RDS file.

Value

A tidylearn_data object (a tibble subclass) with attributes tl_source, tl_format, and tl_timestamp.

Reading files you did not create

readRDS() rebuilds whatever R objects the file describes, so read only files from a source you trust. On R before 4.4.0 a crafted file can run code as it is read (CVE-2024-27322). The remote readers tl_read_github() and tl_read_s3() refuse .rds and .rdata files on those versions unless trust_rds = TRUE.

Examples

```
path <- tempfile(fileext = ".rds")
saveRDS(mtcars, path)
tl_read_rds(path)
unlink(path)
```

`tl_read_s3` *Read from Amazon S3*

Description

Downloads a file from an S3 bucket and reads it into a `tidylearn_data` object. The file format is auto-detected from the key's extension, or can be specified explicitly. Requires the **paws.storage** package and valid AWS credentials.

Usage

```
tl_read_s3(source, format = NULL, region = NULL, ..., trust_rds = FALSE)
```

Arguments

`source`	An S3 URI (e.g., `"s3://bucket/path/to/file.csv"`). Zip archives are not read from S3: download the object and read it with `tl_read_zip()`.
`format`	Optional format override for the downloaded file. If `NULL`, auto-detected from the S3 key extension.
`region`	AWS region. If `NULL`, uses the default from your AWS configuration.
`...`	Additional arguments passed to the format-specific reader.
`trust_rds`	Logical. Read an `.rds`, `.rdata` or `.rda` object on R older than 4.4.0? Those versions can run code embedded in a crafted file as it is read (CVE-2024-27322), so such objects are refused there unless this is `TRUE`. It has no effect on R 4.4.0 or later. Default `FALSE`.

Value

A `tidylearn_data` object containing the downloaded data.

Reading R serialisation files

An `.rds` or `.rdata` object is rebuilt with `readRDS()` or `load()`, which recreate whatever R objects the file describes. Read them only from a bucket you trust, on any version of R.

Examples

```
## Not run:
# Needs AWS credentials
tl_read_s3("s3://my-bucket/data/sales.csv")
tl_read_s3("s3://my-bucket/data/results.parquet", region = "eu-west-1")

## End(Not run)
```

`tl_read_sqlite` *Read from a SQLite database*

Description

Opens a SQLite database file, executes a SQL query, and returns the result as a `tidylearn_data` object. The connection is automatically closed when done. Requires **DBI** and **RSQLite**.

Usage

```
tl_read_sqlite(path, query, ...)
```

Arguments

`path`	Path to a SQLite database file (`.sqlite` or `.db`).
`query`	A SQL query string.
`...`	Additional arguments passed to `DBI::dbGetQuery()`.

Value

A `tidylearn_data` object containing the query results.

Examples

```
# RSQLite imports DBI, so both are available here
path <- tempfile(fileext = ".sqlite")
conn <- DBI::dbConnect(RSQLite::SQLite(), path)
DBI::dbWriteTable(conn, "cars", mtcars)
DBI::dbDisconnect(conn)

tl_read_sqlite(path, "SELECT mpg, cyl, hp FROM cars WHERE cyl = 6")
unlink(path)
```

tl_read_tsv *Read a TSV file*

Description

Reads a tab-separated file into a `tidylearn_data` object. Uses **readr** when available for faster parsing, with a base R fallback.

Usage

```
tl_read_tsv(path, ...)
```

Arguments

`path`	Path to a TSV file.
`...`	Additional arguments passed to `readr::read_tsv()` or `utils::read.delim()`.

Value

A `tidylearn_data` object (a tibble subclass) with attributes `tl_source`, `tl_format`, and `tl_timestamp`.

Examples

```
path <- tempfile(fileext = ".tsv")
write.table(mtcars, path, sep = "\t", row.names = FALSE)
tl_read_tsv(path)
unlink(path)
```

tl_read_zip *Read data from a zip archive*

Description

Extracts a zip archive to a temporary directory and reads the contents. If the archive contains a single data file, it is read directly. If multiple data files are found, they are row-bound with a `source_file` column. Use the `file` argument to select a specific file from the archive.

Usage

```
tl_read_zip(path, file = NULL, format = NULL, .quiet = FALSE, ...)
```

Arguments

path	Path to a zip file.
file	Optional name of a specific file within the archive to read: its path within the archive ("2024/sales.csv"), its file name, or part of its path. An exact path wins, then an exact file name, then a partial match; a name that matches more than one member at the first of those steps that matches anything is an error listing them.
format	Optional format override for the file(s) inside the archive. Without file, this selects the members of that format rather than forcing it onto them, whether the archive holds one data file or several. With file, the named member is read as format whatever its extension.
.quiet	Suppress messages. Default is FALSE.
...	Additional arguments passed to the format-specific reader.

Details

An archive with a member whose name could reach outside the extraction directory – an absolute path, a drive letter on Windows, or any .. component – is refused before anything is extracted.

Value

A tidylearn_data object (a tibble subclass) with attributes tl_source, tl_format, and tl_timestamp. The archive is extracted to a temporary directory that is cleaned up automatically. If multiple data files are found, a source_file column gives each row's member as its path within the archive.

Examples

```
# readr ships a zip archive holding one CSV
archive <- readr::readr_example("mtcars.csv.zip")
tl_read_zip(archive)

# Name a member to read just that one
tl_read_zip(archive, file = "mtcars.csv", .quiet = TRUE)
```

---

tl_reduce_dimensions *Feature Engineering via Dimensionality Reduction*

Description

Use PCA, MDS, or other dimensionality reduction as a preprocessing step for supervised learning. This can improve model performance and interpretability.

Usage

```
tl_reduce_dimensions(
  data,
  response = NULL,
  method = "pca",
  n_components = NULL,
  ...
)
```

Arguments

data	A data frame
response	Response variable name (will be preserved)
method	Dimensionality reduction method: "pca" or "mds"
n_components	Number of components to retain, at most the number the method computes: one per numeric column for PCA, and k (default 2, passed through ...) for MDS. NULL keeps them all.
...	Additional arguments for the dimensionality reduction method

Value

A list with components:

data The transformed data frame with reduced-dimension columns and the response variable (if provided).

reduction_model The fitted tidylearn dimensionality reduction model.

original_data The original input data frame.

response The response variable name, or NULL.

Examples

```
# Reduce dimensions before classification
reduced <- tl_reduce_dimensions(
  iris, response = "Species",
  method = "pca", n_components = 3
)
model <- tl_model(reduced$data, Species ~ ., method = "tree")
```

tl_run_pipeline *Run a tidylearn pipeline*

Description

Run a tidylearn pipeline

Usage

```
tl_run_pipeline(pipeline, verbose = TRUE)
```

Arguments

pipeline	A tidylearn pipeline object
verbose	Logical; whether to print progress

Value

The input tidylearn_pipeline object with its $results component populated. Results include $processed_data (the training data after preprocessing), $preprocessing_stats (the medians, modes, centres and scales learned from the training data, replayed by tl_predict_pipeline), $model_results (a named list of per-model fits and metrics), $best_model_name, $best_model (the winning tidylearn_model), and $metric_values.

The training data is every row when validation = "cv": each model is scored across the folds and then refitted on all of them. With validation = "split" it is the training split alone, since the models kept are the ones fitted there and scored on the test rows.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .,
  models = list(tree = list(method = "tree")),
  evaluation = list(metrics = "accuracy", validation = "cv",
    cv_folds = 2, best_metric = "accuracy"))
pipe <- tl_run_pipeline(pipe, verbose = FALSE)
```

tl_save_pipeline *Save a pipeline to disk*

Description

Save a pipeline to disk

Usage

```
tl_save_pipeline(pipeline, file)
```

Arguments

pipeline	A tidylearn pipeline object
file	Path to save the pipeline

Value

Called for its side effect of saving to disk; returns NULL invisibly.

Examples

```
pipe <- tl_pipeline(iris, Species ~ .)
tl_save_pipeline(pipe, tempfile(fileext = ".rds"))
```

---

tl_semisupervised *Semi-Supervised Learning via Clustering*

Description

Train a supervised model with limited labels by first clustering the data and propagating labels within clusters.

Usage

```
tl_semisupervised(
  data,
  formula,
  labeled_indices,
  cluster_method = "kmeans",
  supervised_method = "tree",
  ...,
  cluster_args = list()
)
```

Arguments

data	A data frame
formula	Model formula. The response must be categorical: a factor, character or logical column, or an expression that computes a factor or character vector from a column, such as factor(am) or factor(mpg > 20). For an expression, each row's propagated label is written to that column as its value in a labelled row of the same class. An expression that then computes other labels, such as cut(mpg, 2), whose breaks follow the column's range, is refused.
labeled_indices	Indices of labeled observations

`cluster_method` Clustering method for label propagation: `"kmeans"` (default), `"pam"`, `"clara"` or `"hclust"`, whose tree is cut at k

`supervised_method`
Supervised learning method for the final model (default: `"tree"`, which handles any number of classes). `"logistic"` is binary-only and errors on a response with more than two levels.

`...` Additional arguments for the supervised model

`cluster_args` A named list of arguments for the clustering step, such as `list(nstart = 5)` for k-means. k is set from the labelled classes and cannot be given here.

Details

Labels are propagated by majority vote within each cluster, so the response must be categorical. The rows are clustered on the predictors the formula names, into as many clusters as the labelled rows hold classes. A labelled row whose label is missing takes no part in the vote. Rows in a cluster where no labelled observation carries a label have no label to take, and labelled rows whose own label is missing have none either; both are left out of training, with a warning giving the counts. The pseudo-labels keep the response's level order, so the second level stays the positive class.

Value

A tidylearn model object with additional class `"tidylearn_semisupervised"`, trained on pseudo-labeled data. The model includes a `semisupervised_info` element with `labeled_indices`, `cluster_model`, `label_mapping`, and `n_unlabelled_dropped`, the number of rows left out for having no label to train on.

Examples

```
# Use only 10% of labels
labeled_idx <- sample(nrow(iris), size = 15)
model <- tl_semisupervised(iris, Species ~ ., labeled_indices = labeled_idx,
  cluster_method = "kmeans",
  supervised_method = "tree"
)
```

---

tl_split *Split data into train and test sets*

Description

Split data into train and test sets

Usage

```
tl_split(data, prop = 0.8, stratify = NULL, seed = NULL)
```

Arguments

data	A data frame
prop	Proportion for training set (default: 0.8)
stratify	Column name for stratified splitting. Each stratum is split at prop. A numeric column with more than five distinct values is stratified by its quartiles, and rows missing the value form a stratum of their own. A stratum of a single row goes to the training set, unless such strata together hold more than a tenth of the rows, as in an ID column; then they are pooled into one stratum and split together.
seed	Random seed for reproducibility

Value

A list with two elements:

$train A data frame containing the training subset.

$test A data frame containing the test subset.

A split that leaves the test set empty, as a single row does, warns.

Examples

```
split_data <- tl_split(iris, prop = 0.7, stratify = "Species")
train <- split_data$train
test <- split_data$test
```

---

tl_step_selection *Perform stepwise selection on a linear model*

Description

Perform stepwise selection on a linear model

Usage

```
tl_step_selection(
  data,
  formula,
  direction = "backward",
  criterion = "AIC",
  trace = FALSE,
  steps = 1000,
  ...
)
```

Arguments

data	A data frame containing the training data
formula	A formula specifying the initial model
direction	Direction of stepwise selection: "forward", "backward", or "both"
criterion	Criterion for selection: "AIC" or "BIC"
trace	Logical; whether to print progress
steps	Maximum number of steps to take
...	Additional arguments to pass to step()

Details

Every candidate model is fitted on the same rows: those with no missing value in any variable of `formula`. When that leaves out rows a candidate could otherwise use, a message gives the count, and the fit records the rows in its `na.action`, as `lm()` records the rows it drops itself.

Forward and both-direction selection start from a model holding the intercept, or none if `formula` removes it with `- 1`, and any `offset()` terms in `formula`. Neither is ever dropped.

Value

A `tidylearn_model` object of class `tidylearn_linear` wrapping the selected `lm` model. Access the underlying model via `$fit` and the selected formula via `$spec$formula`. `$data` is the data passed in.

Examples

```
model <- tl_step_selection(mtcars, mpg ~ ., direction = "backward")
summary(model)
```

---

tl_stratified_models *Stratified Features via Clustering*

Description

Create cluster-specific supervised models for heterogeneous data

Usage

```
tl_stratified_models(
  data,
  formula,
  cluster_method = "kmeans",
  k = 3,
  supervised_method = "tree",
  ...,
  cluster_args = list()
)
```

Arguments

`data`	A data frame
`formula`	Model formula
`cluster_method`	Clustering method: `"kmeans"` (default), `"pam"`, `"clara"` or `"hclust"`, whose tree is cut at `k`. Only k-means can assign new rows, so the others predict their training data alone.
`k`	Number of clusters
`supervised_method`	Supervised learning method (default: `"tree"`, which handles both regression and classification). `"linear"` needs a numeric response and refuses a factor.
`...`	Additional arguments for the supervised models
`cluster_args`	A named list of arguments for the clustering step, such as `list(nstart = 5)` for k-means. Pass k as `k`.

Details

The rows are clustered on the predictors the formula names, and a model is fitted to each cluster. A cluster whose rows all hold one class has nothing for a classifier to separate, so it gets no model and its rows are predicted as that class.

Value

A list with class `"tidylearn_stratified"` containing:

cluster_model The fitted clustering model.

clusters The training rows' cluster assignments.

supervised_models Named list of tidylearn models, one per cluster that holds more than one class.

single_class_clusters Named character vector giving, for each cluster whose rows all hold one class, that class.

formula The model formula.

data The original training data.

Examples

```
models <- tl_stratified_models(mtcars, mpg ~ ., cluster_method = "kmeans",
                               k = 3, supervised_method = "linear")
```

tl_table *Create formatted tables for tidylearn models*

Description

Dispatches to the appropriate table function based on model type and requested table type. Requires the gt package.

Usage

```
tl_table(model, type = "auto", ...)
```

Arguments

model	A tidylearn model object
type	Table type (default: "auto"). For supervised models: "metrics", "coefficients", "confusion", "importance". For unsupervised models: "variance", "loadings", "clusters". MDS models are not supported.
...	Additional arguments passed to the underlying table function

Value

A gt table object.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_table(model)
tl_table(model, type = "coefficients")
```

tl_table_clusters *Formatted cluster summary table*

Description

Produces a styled gt table showing cluster sizes and mean feature values for the columns the clustering used. Supports kmeans, pam, clara, dbscan, and hclust models.

Usage

```
tl_table_clusters(model, k = 3, digits = 2, ...)
```

Arguments

model	A tidylearn clustering model object
k	For hclust models, the number of clusters to cut (default: 3)
digits	Number of decimal places (default: 2)
...	Additional arguments (currently unused)

Value

A gt table object.

Examples

```
model <- tl_model(iris[, 1:4], method = "kmeans", k = 3)
tl_table_clusters(model)
```

---

tl_table_coefficients *Formatted model coefficients table*

Description

Produces a styled gt table of model coefficients. Supports linear, polynomial, logistic, ridge, lasso, and elastic net models. The numbers come from tl_coefficients, which returns them as a tibble if you would rather format them yourself.

Usage

```
tl_table_coefficients(
  model,
  lambda = "1se",
  digits = 4,
  conf_int = FALSE,
  level = 0.95,
  exponentiate = FALSE,
  ...
)
```

Arguments

model	A tidylearn model object
lambda	For regularised models: "1se" (default), "min", or a numeric penalty within the fitted path
digits	Number of decimal places (default: 4)

conf_int	Whether to add a confidence interval (default: FALSE). Not available for regularised models.
level	Confidence level for the interval (default: 0.95)
exponentiate	Whether to report odds ratios rather than log odds (default: FALSE). Classification models only.
...	Additional arguments (currently unused)

Value

A gt table object.

See Also

tl_coefficients for the underlying tibble.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_table_coefficients(model)
tl_table_coefficients(model, conf_int = TRUE)
```

---

tl_table_comparison *Compare multiple models in a formatted table*

Description

Evaluates multiple tidylearn models and presents the results side-by-side in a styled gt table.

Usage

```
tl_table_comparison(..., new_data = NULL, names = NULL, digits = 4)
```

Arguments

...	tidylearn model objects to compare
new_data	Optional test data for evaluation. If NULL, the models are scored on their training data, which they must share: models fitted on different data are an error asking for new_data. A model fitted on engineered features, as tl_auto_ml() builds some of its candidates, is scored on the training data of the others.
names	Optional character vector of model names
digits	Number of decimal places (default: 4)

Value

A gt table object. Its source note counts the rows scored, per model when the models scored different rows.

Examples

```
m1 <- tl_model(mtcars, mpg ~ ., method = "linear")
m2 <- tl_model(mtcars, mpg ~ ., method = "lasso")
tl_table_comparison(m1, m2, names = c("Linear", "Lasso"))
```

---

tl_table_confusion *Formatted confusion matrix table*

Description

Produces a styled gt confusion matrix with correct predictions highlighted. Only available for classification models.

Usage

```
tl_table_confusion(model, new_data = NULL, ...)
```

Arguments

model	A tidylearn classification model
new_data	Optional test data. If NULL, uses training data.
...	Additional arguments (currently unused)

Value

A gt table object, with a row and a column for each class the model was trained on. Rows of a class the model never saw are left out, with a warning.

Examples

```
model <- tl_model(iris, Species ~ ., method = "forest")
tl_table_confusion(model)
```

tl_table_importance *Formatted feature importance table*

Description

Produces a styled gt table of feature importance with a colour gradient. Supports tree-based, regularised, and xgboost models.

Usage

```
tl_table_importance(model, top_n = 20, digits = 2, ...)
```

Arguments

model	A tidylearn model object
top_n	Maximum number of features to display (default: 20)
digits	Number of decimal places (default: 2)
...	Additional arguments (currently unused)

Value

A gt table object.

Examples

```
model <- tl_model(iris, Species ~ ., method = "forest")
tl_table_importance(model)
```

tl_table_loadings *Formatted PCA loadings table*

Description

Produces a styled gt table of variable loadings on each principal component, with a diverging colour scale to highlight strong loadings.

Usage

```
tl_table_loadings(model, n_components = NULL, digits = 3, ...)
```

Arguments

model	A tidylearn PCA model object
n_components	Number of components to show (default: all)
digits	Number of decimal places (default: 3)
...	Additional arguments (currently unused)

Value

A gt table object.

Examples

```
model <- tl_model(iris[, 1:4], method = "pca")
tl_table_loadings(model)
```

---

tl_table_metrics *Formatted evaluation metrics table*

Description

Produces a styled gt table of model evaluation metrics from tl_evaluate.

Usage

```
tl_table_metrics(model, new_data = NULL, digits = 4, ...)
```

Arguments

model	A tidylearn supervised model object
new_data	Optional test data. If NULL, uses training data.
digits	Number of decimal places (default: 4)
...	Additional arguments passed to tl_evaluate

Value

A gt table object. Its source note counts the rows scored.

Examples

```
model <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
tl_table_metrics(model)
```

tl_table_variance *Formatted PCA variance explained table*

Description

Produces a styled gt table of variance explained by each principal component, with a colour gradient on cumulative variance.

Usage

```
tl_table_variance(model, n_components = NULL, digits = 4, ...)
```

Arguments

model	A tidylearn PCA model object
n_components	Maximum number of components to show (default: all)
digits	Number of decimal places (default: 4)
...	Additional arguments (currently unused)

Value

A gt table object.

Examples

```
model <- tl_model(iris[, 1:4], method = "pca")
tl_table_variance(model)
```

tl_test_interactions *Test for significant interactions between variables*

Description

Test for significant interactions between variables

Usage

```
tl_test_interactions(
  data,
  formula,
  var1 = NULL,
  var2 = NULL,
  all_pairs = FALSE,
  categorical_only = FALSE,
  numeric_only = FALSE,
  mixed_only = FALSE,
  alpha = 0.05
)
```

Arguments

data	A data frame containing the data
formula	A formula specifying the base model without interactions, or a string that parses as one. . and - var are expanded against data. The models are fitted with lm(), so the response must be numeric or logical.
var1	First variable to test for interactions
var2	Second variable to test for interactions (if NULL, tests var1 with all others)
all_pairs	Logical; whether to test all variable pairs
categorical_only	Logical; whether to only test categorical variables
numeric_only	Logical; whether to only test numeric variables
mixed_only	Logical; whether to only test numeric-categorical pairs
alpha	Significance level for interaction tests

Value

A data frame with one row per tested interaction pair, containing columns var1, var2, p_value, significant (logical), delta_r2 (change in R-squared), and f_statistic, sorted by p_value ascending.

Examples

```
results <- tl_test_interactions(mtcars, mpg ~ wt + hp + cyl,
  var1 = "wt", var2 = "hp")
```

`tl_test_model_difference` *Perform statistical comparison of models using cross-validation*

Description

Perform statistical comparison of models using cross-validation

Usage

```
tl_test_model_difference(
  cv_results,
  baseline_model = NULL,
  test = "t.test",
  metric = NULL
)
```

Arguments

`cv_results`	Results from tl_compare_cv function
`baseline_model`	Name of the model to use as baseline for comparison
`test`	Type of statistical test: `"t.test"`, a paired t-test, or `"wilcox"`, a paired Wilcoxon signed-rank test (`"wilcox.test"` is read as `"wilcox"`). With no tied or zero differences the signed-rank test is exact, and the smallest two-sided p-value it can give for `n` folds is `2 / 2^n`: 0.0625 for 5 folds, so it needs at least 6 to reach `p < 0.05`.
`metric`	Name of the metric to compare

Value

A data frame with columns `metric`, `model`, `baseline`, `mean_diff`, `p_value`, and `p_adj` (Holm-adjusted p-value) containing pairwise statistical comparisons against the baseline model. Each comparison pairs the folds both models have a value for, and `mean_diff` is the mean of those paired differences. With fewer than two such folds the p-values are `NA`, with a warning.

Examples

```
m1 <- tl_model(mtcars, mpg ~ wt, method = "linear")
m2 <- tl_model(mtcars, mpg ~ wt + hp, method = "linear")
cv <- tl_compare_cv(mtcars, list(simple = m1, full = m2), folds = 3)
tl_test_model_difference(cv, baseline_model = "simple", metric = "rmse")
```

`tl_transfer_learning` *Transfer Learning Workflow*

Description

Use unsupervised pre-training before supervised learning: the predictors the formula names are projected onto their principal components, and the supervised model is fitted on the component scores.

Usage

```
tl_transfer_learning(
  data,
  formula,
  pretrain_method = "pca",
  supervised_method = "tree",
  ...
)
```

Arguments

`data`	Training data
`formula`	Model formula, or a string that parses as one. The PCA is fitted on the numeric predictors it names.
`pretrain_method`	Pre-training method. Only `"pca"` is available: `predict()` has to project new rows, and PCA is the reduction that can.
`supervised_method`	Supervised learning method (default: `"tree"`, which handles both regression and classification with any number of classes). `"logistic"` is binary-only and errors on a response with more than two levels.
`...`	Additional arguments passed to `tl_reduce_dimensions`, such as `n_components`

Value

A list with class `"tidylearn_transfer"` containing:

pretrain_model The fitted dimensionality reduction model.

supervised_model The fitted supervised tidylearn model.

formula The model formula.

method The supervised learning method used.

Examples

```
model <- tl_transfer_learning(iris, Species ~ .,
  pretrain_method = "pca", supervised_method = "tree")
```

`tl_tune_deep` *Tune a deep learning model*

Description

Tune a deep learning model

Usage

```
tl_tune_deep(
  data,
  formula,
  is_classification = NULL,
  hidden_layers_options = list(c(32), c(64, 32), c(128, 64, 32)),
  learning_rates = c(0.01, 0.001, 1e-04),
  batch_sizes = c(16, 32, 64),
  epochs = 30,
  validation_split = 0.2,
  ...
)
```

Arguments

`data`	A data frame containing the training data
`formula`	A formula specifying the model
`is_classification`	Logical indicating if this is a classification problem. `NULL` (default) reads it from the response, as `tl_model` does: a factor or character response is classification. `FALSE` with such a response is an error.
`hidden_layers_options`	List of vectors defining hidden layer configurations to try
`learning_rates`	Learning rates to try (default: c(0.01, 0.001, 0.0001))
`batch_sizes`	Batch sizes to try (default: c(16, 32, 64))
`epochs`	Number of training epochs (default: 30)
`validation_split`	Proportion of the rows held out to score each configuration on, drawn at random (default: 0.2). Every configuration is scored on the same rows.
`...`	Additional arguments passed to keras's fit() for every configuration; `verbose` (default 0) replaces the value used otherwise. Arguments with one value per row – `weights`, `subset`, `offset`, `foldid`, `strata` – are refused, since each configuration is fitted on part of the rows.

Value

A list with elements `model` (the best configuration refitted as a `tidylearn_model`, so `predict()` and the deep plots take it; the keras model is at `$model$fit$model`), `best_hidden_layers` (optimal layer configuration), `best_learning_rate`, `best_batch_size`, and `tuning_results` (a data frame of all hyperparameter combinations and their validation losses).

Examples

```
## Not run:
if (requireNamespace("keras3", quietly = TRUE)) {
  result <- tl_tune_deep(iris, Species ~ .,
    hidden_layers_options = list(c(10), c(10, 5)),
    learning_rates = c(0.01, 0.001), batch_sizes = c(32),
    epochs = 5)
  predict(result$model, iris[1:5, ])
}

## End(Not run)
```

`tl_tune_grid` *Tune hyperparameters for a model using grid search*

Description

Tune hyperparameters for a model using grid search

Usage

```
tl_tune_grid(
  data,
  formula,
  method,
  param_grid,
  folds = 5,
  metric = NULL,
  maximize = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`data`	A data frame containing the training data
`formula`	A formula specifying the model
`method`	The modeling method to tune, one of the supervised methods `tl_model` fits
`param_grid`	A named list of candidate values, one element for each `tl_model` argument to tune, named after it

folds	Number of cross-validation folds, a whole number between 2 and nrow(data). nrow(data) leaves each row out in turn, and each fold then scores a single prediction. "accuracy", "mae", "mse" and "mape" average to their values over the left-out predictions. The average "rmse" is the mean absolute error; "precision", "recall", "sensitivity", "specificity" and "f1" are undefined on the folds whose one row gives them nothing to divide by; and "rsq", "auc" and "pr_auc" are undefined on every fold. A run scoring any of these warns once.
metric	Metric to optimize: one of the names tl_evaluate computes for the task. Defaults to "accuracy" for classification and "rmse" for regression.
maximize	Logical; whether to maximize (TRUE) or minimize (FALSE) the metric. NULL, the default, maximizes every metric but the error metrics "rmse", "mse", "mae" and "mape".
verbose	Logical; whether to print progress
...	Additional arguments passed to tl_model for every fold and for the final fit. Arguments holding one value per row of data – weights, subset, offset, foldid and strata – are refused, since they cannot follow the rows into a fold.

Value

A tidylearn model object fitted with the best hyperparameters. Tuning results are stored as an attribute "tuning_results", a list containing param_grid, results, best_params, best_metric, metric, and maximize.

results has one row per evaluated combination: mean_metric (the mean over the folds that produced a score), n_folds_ok (how many of the folds did), and a column per parameter. A parameter with a vector-valued candidate, such as hidden_layers, is a list column. A fold produces no score when its fit fails, when the metric is undefined on it – "auc" on a fold holding one class, or "precision" on one where nothing is predicted positive – or when none of its rows can be scored, as when every predictor is missing there, which is warned about.

Only combinations with n_folds_ok equal to folds are eligible to be best, since a mean over the folds that happened to succeed is not comparable with a mean over all of them. If no combination was scored on every fold, the best of those scored on the most folds is used, with a warning saying whether fits failed or the metric was undefined. If no combination was scored on any fold, the function stops.

For method = "forest", an mtry above the number of predictors is capped at that number, with a warning, and duplicate combinations that result are evaluated once. Predictors are counted as the forest is fitted on them, so a column removed with - id is not one, and a matrix-valued term such as poly(hp, 2) is one per column.

Examples

```
model <- tl_tune_grid(iris, Species ~ ., method = "tree",
  param_grid = list(cp = c(0.01, 0.1), minsplit = c(10, 20)),
  folds = 2, verbose = FALSE)
```

`tl_tune_nn` *Tune a neural network model*

Description

Tune a neural network model

Usage

```
tl_tune_nn(
  data,
  formula,
  is_classification = NULL,
  sizes = c(1, 2, 5, 10),
  decays = c(0, 0.001, 0.01, 0.1),
  folds = 5,
  ...
)
```

Arguments

`data`	A data frame containing the training data
`formula`	A formula specifying the model
`is_classification`	Logical indicating if this is a classification problem. `NULL` (default) reads it from the response, as `tl_model` does: a factor or character response is classification. `FALSE` with such a response is an error.
`sizes`	Vector of hidden layer sizes to try
`decays`	Vector of weight decay parameters to try
`folds`	Number of cross-validation folds (default: 5)
`...`	Additional arguments to pass to nnet(). `maxit` (default 100) and `trace` (default `FALSE`) replace the values used otherwise. Arguments with one value per row – `weights`, `subset`, `offset`, `foldid`, `strata` – are refused, since each fold fits a subset of the rows.

Value

A list with elements `model` (the best fitted `nnet` model), `best_size` (optimal hidden-layer size), `best_decay` (optimal weight decay), and `tuning_results` (a data frame of all parameter combinations and their cross-validated errors: the misclassification rate for classification, the mean squared error for regression).

Examples

```
tuned <- tl_tune_nn(iris, Species ~ .,
  is_classification = TRUE,
  sizes = c(2, 5), decays = c(0, 0.01), folds = 3)

tuned$best_size
tuned$best_decay
tuned$tuning_results

# The grid this searched, drawn as a heatmap
tl_plot_nn_tuning(tuned)
```

`tl_tune_random` *Tune hyperparameters using random search*

Description

Tune hyperparameters using random search

Usage

```
tl_tune_random(
  data,
  formula,
  method,
  param_space,
  n_iter = 10,
  folds = 5,
  metric = NULL,
  maximize = NULL,
  verbose = TRUE,
  seed = NULL,
  ...
)
```

Arguments

`data`	A data frame containing the training data
`formula`	A formula specifying the model
`method`	The modeling method to tune, one of the supervised methods `tl_model` fits
`param_space`	A named list of parameter spaces to sample from, one element for each `tl_model` argument to tune, named after it. Each element is read by its type and length: **a function** called with no arguments to draw one value **a list** a set of candidates, each drawn whole, e.g. `list(c(10), c(20, 10))` for `hidden_layers`

`c(min, max, "log")` log-uniform draw between `min` and `max`

a single value used as given in every iteration

two whole numbers integer range, e.g. `c(10, 20)` draws from 10:20

three or more numbers a discrete set, sampled from as given, whether or not they are whole

two other numbers uniform draw between them, e.g. `c(0.01, 0.1)`

character or factor categorical, sampled from as given

logical sampled from the values given

`n_iter` Number of random parameter combinations to try, a whole number of at least 1

`folds` Number of cross-validation folds, a whole number between 2 and `nrow(data)`. `nrow(data)` leaves each row out in turn, and each fold then scores a single prediction. `"accuracy"`, `"mae"`, `"mse"` and `"mape"` average to their values over the left-out predictions. The average `"rmse"` is the mean absolute error; `"precision"`, `"recall"`, `"sensitivity"`, `"specificity"` and `"f1"` are undefined on the folds whose one row gives them nothing to divide by; and `"rsq"`, `"auc"` and `"pr_auc"` are undefined on every fold. A run scoring any of these warns once.

`metric` Metric to optimize, as for `tl_tune_grid`

`maximize` Logical; whether to maximize (TRUE) or minimize (FALSE) the metric. `NULL`, the default, follows the metric, as for `tl_tune_grid`.

`verbose` Logical; whether to print progress

`seed` Random seed for reproducibility

`...` Additional arguments passed to `tl_model` for every fold and for the final fit. Per-row arguments are refused, as for `tl_tune_grid`.

Value

A tidylearn model object fitted with the best hyperparameters. Tuning results are stored as an attribute `"tuning_results"`, a list containing `param_space`, `results`, `best_params`, `best_metric`, `metric`, and `maximize`.

`results` has one row per iteration: `iteration`, `mean_metric`, `n_folds_ok`, and a column per parameter, as described for `tl_tune_grid`. The best parameters are chosen by the same rules, and `mtry` is capped the same way; duplicate draws are kept, so there are always `n_iter` rows.

Examples

```
model <- tl_tune_random(mtcars, mpg ~ ., method = "tree",
  param_space = list(cp = c(0.01, 0.1), minsplit = c(10, 20)),
  n_iter = 3, folds = 2, verbose = FALSE)
```

`tl_tune_xgboost` *Tune XGBoost hyperparameters*

Description

Tune XGBoost hyperparameters

Usage

```
tl_tune_xgboost(
  data,
  formula,
  is_classification = NULL,
  param_grid = NULL,
  cv_folds = 5,
  nrounds = 1000,
  early_stopping_rounds = 10,
  verbose = TRUE,
  ...
)
```

Arguments

`data`	A data frame containing the training data
`formula`	A formula specifying the model
`is_classification`	Logical indicating if this is a classification problem. `NULL` (default) reads it from the response, as `tl_model` does: a factor or character response is classification. `FALSE` with such a response is an error.
`param_grid`	Named list of parameter values to try. NULL (default) tries every combination of `tl_default_param_grid("xgboost", size = "large")` without its `nrounds`, which early stopping chooses here.
`cv_folds`	Number of cross-validation folds (default: 5)
`nrounds`	Upper bound on boosting rounds per parameter set (default: 1000). Early stopping normally halts well short of it, so this is a ceiling rather than a target; lower it to cap the search.
`early_stopping_rounds`	Early stopping rounds (default: 10)
`verbose`	Logical indicating whether to print progress (default: TRUE)
`...`	Arguments `xgboost::xgb.cv()` takes, such as `showsd` or `stratified`, which go to it alone; case `weights`, one per row of `data`, which the folds split with the rows; and booster parameters held fixed across the grid, such as `nthread` or `tree_method`, which join each parameter set and the final fit. A value in `param_grid` replaces one given here. The other per-row arguments – `subset`, `offset`, `foldid`, `strata` – are refused.

Value

A tidylearn_model object (the refit on full data using the best hyperparameters, built by tl_model so that it records them in $spec$args) with an attribute "tuning_results" containing a list with elements param_grid, results (per-combination CV output), best_params, best_iteration, best_score, and minimize.

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  # The default grid is every combination of the large xgboost grid
  # without nrounds -- this many:
  default_grid <- tl_default_param_grid("xgboost", size = "large")
  prod(lengths(default_grid[names(default_grid) != "nrounds"]))

  # Name a smaller one to see it run, and cap nrounds so early stopping
  # has less ground to cover. nthread = 2 keeps xgboost from taking
  # every core it is offered.
  tuned <- tl_tune_xgboost(iris, Species ~ .,
    param_grid = list(max_depth = c(2, 4)),
    cv_folds = 3, nrounds = 20, verbose = FALSE, nthread = 2)

  results <- attr(tuned, "tuning_results")
  results$best_params
  results$best_iteration

  # tuned is an ordinary model, refit on all rows at those settings
  predict(tuned, iris[1:5, ])
}
```

---

tl_version *Get tidylearn version information*

Description

Get tidylearn version information

Usage

```
tl_version()
```

Value

A package_version object containing the version number

Examples

```
tl_version()
```

tl_xgboost_shap *Generate SHAP values for XGBoost model interpretation*

Description

Generate SHAP values for XGBoost model interpretation

Usage

```
tl_xgboost_shap(model, data = NULL, n_samples = 100, trees_idx = NULL)
```

Arguments

model	A tidylearn XGBoost model object
data	Data for SHAP value calculation (default: NULL, uses training data). The response column is not needed.
n_samples	Number of samples to use (default: 100, NULL for all)
trees_idx	Boosting rounds to include, as a run of consecutive rounds counted from 1 such as 1:20 (default: NULL, uses every round)

Value

A data frame with one column of SHAP values per feature (the columns of the model's design matrix), a BIAS column and a row_id column giving the row of the (sampled) data each row explains. For a multiclass model there is one block of rows per class, told apart by a class column. The columns of data whose names are not already taken – the response, and any factor predictor, whose SHAP columns are named after its levels – are appended for reference.

Examples

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  model <- tl_model(mtcars, mpg ~ ., method = "xgboost", nthread = 2)
  shap <- tl_xgboost_shap(model, n_samples = 20)
}
```

visualize_rules *Visualize Association Rules*

Description

Create visualizations of association rules

Usage

```
visualize_rules(rules_obj, method = "scatter", top_n = 50, ...)
```

Arguments

rules_obj	A tidy_apriori object or an arules rules object. A table of rules is refused: the plots need the rules object.
method	Visualization method: "scatter" (default), drawn by tidylearn, or a method of **arulesViz**'s `plot()`, such as "graph", "grouped", "matrix" or "paracoord"
top_n	Number of rules to visualize, those with the highest lift (default: 50)
...	Additional arguments passed to plot() for rules visualization

Value

For `method = "scatter"`, a ggplot object. Other methods return what **arulesViz**'s `plot()` returns: a ggplot object for "graph", "grouped" and "matrix", and for "paracoord", which draws with grid, a grid `vpPath`.

Examples

```
if (requireNamespace("arules", quietly = TRUE)) {
  data("Groceries", package = "arules")
  res <- tidy_apriori(Groceries, support = 0.001, confidence = 0.5)
  visualize_rules(res, method = "scatter")
}
```

Index

apriori, *46*
augment_dbscan, 7
augment_hclust, 7
augment_kmeans, 8
augment_pam, 9
augment_pca, 9

calc_validation_metrics, 10, *12*, *94*
calc_wss, 11, *18*
clara, *47*
clusGap, *51*
cmdscale, *57*
compare_clusterings, 11
compare_distances, 12
confint, *84*
create_cluster_dashboard, 13

daisy, *52*
dist, *12*, *48*, *50*, *52*, *56*

explore_dbscan_params, 14

filter_rules_by_item, 14
find_related_items, 15

get_pca_loadings, 16
get_pca_variance, 16
ggplot, *13*, *20–22*, *24–28*, *62*, *86*, *92*, *103–116*, *118–121*, *168*
grid.arrange, *13*, *21*, *93*
grob, *93*
gt, *149–155*
gtable, *21*

hclust, *23*, *49*

inspect_rules, 17
isoMDS, *57*

kNNdistplot, *43*

lm, *147*

maxSE, *51*
mds, *59*

optimal_clusters, 18
optimal_hclust_k, 18

pam, *60*
plot.tidylearn_eda, 19
plot.tidylearn_model, 20
plot_cluster_comparison, 20
plot_cluster_sizes, 21
plot_clusters, 22
plot_dendrogram, 23
plot_distance_heatmap, 23
plot_elbow, 24
plot_gap_stat, 25
plot_knn_dist, 25
plot_mds, 26
plot_silhouette, 27
plot_variance_explained, 27
plotnet, *112*
predict.tidylearn_model, 28
predict.tidylearn_stratified, 29
predict.tidylearn_transfer, 30
print.tidy_apriori, 30
print.tidy_dbscan, 31
print.tidy_gap, 32
print.tidy_hclust, 32
print.tidy_kmeans, 33
print.tidy_mds, 34
print.tidy_pam, 34
print.tidy_pca, 35
print.tidy_silhouette, 36
print.tidylearn_automl, 36
print.tidylearn_compute_advice, 37
print.tidylearn_data, 38
print.tidylearn_eda, 38
print.tidylearn_gpu_check, 39
print.tidylearn_model, 40
print.tidylearn_pipeline, 40

recommend_products, 41
rpart.plot, *117*

sammon, *58*
shinyApp, *90*
standardize_data, 42
suggest_eps, 42
summarize_rules, 43
summary.tidylearn_model, 44
summary.tidylearn_pipeline, 45

tibble, *28–30*, *78*, *90*, *94*, *123*, *125*, *127*, *130*, *132*, *135–137*, *140, 141*
tidy_apriori, 45
tidy_clara, 47
tidy_cutree, 48
tidy_dbscan, *10*, *43*, 48
tidy_dendrogram, 49
tidy_dist, 50
tidy_gap_stat, *18*, 51
tidy_gower, 52, *56*
tidy_hclust, 53
tidy_kmeans, 54
tidy_knn_dist, 55
tidy_mds, 55
tidy_mds_classical, 56
tidy_mds_kruskal, 57
tidy_mds_sammon, 58
tidy_mds_smacof, 59
tidy_pam, *47*, 59
tidy_pca, *16*, 60
tidy_pca_biplot, 61
tidy_pca_screeplot, 62
tidy_rules, *14, 15*, *41*, *46*, 63
tidy_silhouette, 64, *94*
tidy_silhouette_analysis, *18*, 64
tidylearn-classification, 65
tidylearn-cloud-consent, 65
tidylearn-cloud-cost, 66
tidylearn-cloud-endpoint, 66
tidylearn-cloud-serialize, 66
tidylearn-coefficients, 66
tidylearn-core, 67
tidylearn-deep-learning, 67
tidylearn-diagnostics, 67
tidylearn-interactions, 67
tidylearn-metrics, 68
tidylearn-model-selection, 68
tidylearn-neural-networks, 68
tidylearn-pipeline, 68
tidylearn-read, 69
tidylearn-read-backends, 70
tidylearn-regression, 70
tidylearn-regularization, 71
tidylearn-svm, 71
tidylearn-tables, 71
tidylearn-trees, 71
tidylearn-tuning, 71
tidylearn-visualization, 72
tidylearn-workflows, 72
tidylearn-xgboost, 72
tl_add_cluster_features, 72
tl_anomaly_aware, 73, *85*
tl_auto_interactions, 74
tl_auto_ml, 75, *85*
tl_calc_classification_metrics, 77, *94*
tl_check_assumptions, 79
tl_check_gpu, 79
tl_cloud_allow_host, 80
tl_cloud_allow_host(), *81, 82*
tl_cloud_allowed_hosts, 81
tl_cloud_allowed_hosts(), *81*
tl_cloud_consent, 82
tl_cloud_jobs, 83
tl_coefficients, 83, *150, 151*
tl_compare_cv, 85
tl_compare_pipeline_models, 86
tl_compute_advisor, 87, *100*
tl_cv, 89, *94*
tl_dashboard, 90
tl_default_param_grid, 91
tl_detect_outliers, 92
tl_diagnostic_dashboard, 93
tl_evaluate, *78*, *85*, *89*, 94, *154*, *161*
tl_explore, 95
tl_get_best_model, 96
tl_influence_measures, 96
tl_interaction_effects, 97
tl_load_pipeline, 99
tl_model, *16*, *84, 85*, *89*, 99, *159–166*
tl_model(), *88*
tl_pipeline, 102, *124*
tl_plot_cv_comparison, 103
tl_plot_cv_results, 104
tl_plot_deep_architecture, 104
tl_plot_deep_history, 105
tl_plot_gain, 106

tl_plot_importance_comparison, 106
tl_plot_importance_regularized, 107
tl_plot_influence, 108
tl_plot_interaction, 109
tl_plot_intervals, 110
tl_plot_lift, 111
tl_plot_model_comparison, 111
tl_plot_nn_architecture, 112
tl_plot_nn_tuning, 113
tl_plot_partial_dependence, 113
tl_plot_regularization_cv, 114
tl_plot_regularization_path, 115
tl_plot_svm_boundary, 115
tl_plot_svm_tuning, 116
tl_plot_tree, 117
tl_plot_tuning_results, 118
tl_plot_xgboost_importance, 119
tl_plot_xgboost_shap_dependence, 120
tl_plot_xgboost_shap_summary, 121
tl_plot_xgboost_tree, 121
tl_predict_pipeline, *96*, 122, *143*
tl_prepare_data, 123
tl_read, 125
tl_read_bigquery, 126
tl_read_csv, 127
tl_read_db, 128
tl_read_dir, 128
tl_read_excel, 130
tl_read_github, 130
tl_read_json, 131
tl_read_kaggle, 132
tl_read_mysql, 133
tl_read_parquet, 134
tl_read_postgres, 135
tl_read_rdata, 136
tl_read_rds, 137
tl_read_s3, 138
tl_read_sqlite, 139
tl_read_tsv, 140
tl_read_zip, 140
tl_reduce_dimensions, 141, *158*
tl_run_pipeline, *103*, 143
tl_save_pipeline, *99*, 143
tl_semisupervised, *85*, 144
tl_split, 145
tl_step_selection, 146
tl_stratified_models, 147
tl_table, 149
tl_table_clusters, 149
tl_table_coefficients, *84*, 150
tl_table_comparison, 151
tl_table_confusion, 152
tl_table_importance, *83*, 153
tl_table_loadings, 153
tl_table_metrics, 154
tl_table_variance, 155
tl_test_interactions, *75*, 155
tl_test_model_difference, 157
tl_transfer_learning, 158
tl_tune_deep, 159
tl_tune_grid, *91*, 160, *164*
tl_tune_nn, *113*, 162
tl_tune_random, *91*, 163
tl_tune_xgboost, *91*, 165
tl_version, 166
tl_xgboost_shap, 167

visualize_rules, 167

xgb.plot.tree, *122*