Package 'featForge'

April 16, 2025

Title Automated Feature Engineering for Credit Scoring

Version 0.1.2

Maintainer Rudolfs Kregers <rudolfs.kregers@gmail.com>

Description Automated feature engineering functions tailored for credit scoring. It includes utilities for extracting structured features from timestamps, IP addresses, and email addresses, enabling enhanced predictive modeling for financial risk assessment.

Depends R (>= 3.5.0)

Imports stats, utils

Suggests testthat (>= 3.0.0), stringdist, Matrix

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Author Rudolfs Kregers [aut, cre]

Repository CRAN

Date/Publication 2025-04-16 14:20:05 UTC

Contents

`aggregate_applications`

Aggregate Numeric Data by Periods

Description

Aggregates any numeric variable(s) in a dataset over defined time periods and returns summary features computed from provided operation functions. E.g., aggregating and making features from transactional data, previous loan repayment behavior, credit bureau inquiries. Aggregation is performed by a specified grouping identifier (e.g., application, client, or agreement level) and is based on time-periods.

Usage

```
aggregate_applications(
  data,
  id_col,
  amount_col,
  time_col = NULL,
  group_cols = NULL,
  ops,
  period,
  observation_window_start_col = NULL,
  scrape_date_col = NULL,
  period_agg = sum,
  period_missing_inputs = 0
)
```

Arguments

`data`	A data frame containing the data to be aggregated. The dataset must include at least the columns specified by `id_col`, `time_col`, and `amount_col` (or any numeric variable to aggregate).
`id_col`	A character string specifying the column name used to define the aggregation level (e.g., `"application_id"`, `"client_id"`, or `"agreement_id"`).
`amount_col`	A character string specifying the column in `data` that contains the numeric variable to be aggregated. This variable can represent transaction amounts, loan repayment values, credit bureau inquiry counts, or any other numeric measure.
`time_col`	A character string indicating the column name that contains the date (or timestamp) when the event occurred. This column must be of class `Date` or `POSIXct`.

`group_cols` An optional character vector of column names by which to further subdivide the aggregation. For each unique value in these columns, separate summary features will be generated and appended as new columns.

`ops` A named list of functions used to compute summary features on the aggregated period values. Each function must accept a single numeric vector as input. The names of the list elements are used to label the output columns.

`period` Either a character string specifying the time period grouping (`"daily"`, `"weekly"`, `"monthly"`, or `"all"`) or a numeric vector of length 2 (e.g., `c(7, 8)`) where the first element is the cycle length in days and the second is the number of consecutive cycles. When set to `"all"`, the entire set of observations is aggregated as a single period, effectively disabling time aggregation.

`observation_window_start_col`
A character string indicating the column name that contains the observation window start date. This argument is required when `period` is specified as a character string other than `"all"`.

`scrape_date_col`
A character string indicating the column name that contains the scrape date (i.e., the end date for the observation window). This is required when `period` is specified as a character string other than `"all"` or as a numeric vector.

`period_agg` A function used to aggregate the numeric values within each period. The default is `sum`. The argument is ignored if `period` is `"all"`.

`period_missing_inputs`
A numeric constant used to replace missing values in periods with no observed data. The default value is `0`.

Details

When `period` is provided as a character string (one of `"daily"`, `"weekly"`, or `"monthly"`), data are grouped into complete calendar periods. For example, if the scrape date falls mid-month, the incomplete last period is excluded. Alternatively, `period` may be specified as a numeric vector of length 2 (e.g., `c(7, 8)`), in which case the first element defines the cycle length in days and the second element the number of consecutive cycles. In this example, if the scrape date is `"2024-12-31"`, the periods span the last 56 days (8 consecutive 7-day cycles), with the first period starting on `"2024-11-05"`.

`aggregate_applications` aggregates numeric data either by defined time periods or over the full observation window. Data is first grouped by the identifier specified in `id_col` (e.g., at the application, client, or agreement level).

1. When `period` is set to `"daily"`, `"weekly"`, or `"monthly"`, transaction dates in `time_col` are partitioned into complete calendar periods (incomplete periods are excluded).
2. When `period` is set to a numeric vector of length 2 (e.g., `c(7, 8)`), consecutive cycles of fixed length are defined.
3. When `period` is set to `"all"`, time aggregation is disabled. All observations for an identifier (or group) are aggregated together.

For each period, the numeric values in `amount_col` (or any other numeric measure) are aggregated using the function specified by `period_agg`. Then, for each unique group (if any `group_cols`

are provided) and for each application (or other identifier), the summary functions specified in ops are applied to the vector of aggregated period values. When grouping is used, the resulting summary features are appended as new columns with names constructed in the format: <operation>_<group_column>_<group_value Missing aggregated values in periods with no observations are replaced by period_missing_inputs.

Value

A data frame where each row corresponds to a unique identifier (e.g., application, client, or agreement). The output includes aggregated summary features for each period and, if applicable, additional columns for each group defined in group_cols.

Examples

```
data(featForge_transactions)

# Example 1: Aggregate outgoing transactions (amount < 0) on a monthly basis.
aggregate_applications(featForge_transactions[featForge_transactions$amount < 0, ],
                       id_col = 'application_id',
                       amount_col = 'amount',
                       time_col = 'transaction_date',
                       ops = list(
                         avg_momnthly_outgoing_transactions = mean,
                         last_month_transactions_amount = function(x) x[length(x)],
# In the aggregated numeric vector, the last observation represents the most recent period.
                 last_month_transaction_amount_vs_mean = function(x) x[length(x)] / mean(x)
                       ),
                       period = 'monthly',
                       observation_window_start_col = 'obs_start',
                       scrape_date_col = 'scrape_date'
)

# Example 2: Aggregate transactions by category and direction.
featForge_transactions$direction <- ifelse(featForge_transactions$amount > 0, 'in', 'out')
aggregate_applications(featForge_transactions,
                       id_col = 'application_id',
                       amount_col = 'amount',
                       time_col = 'transaction_date',
                       group_cols = c('category', 'direction'),
                       ops = list(
                         avg_monthly_transactions = mean,
                         highest_monthly_transactions_count = max
                       ),
                       period = 'monthly',
                       period_agg = length,
                       observation_window_start_col = 'obs_start',
                       scrape_date_col = 'scrape_date'
)

# Example 3: Aggregate using a custom numeric period:
# 30-day cycles for 3 consecutive cycles (i.e., the last 90 days).
aggregate_applications(featForge_transactions,
                       id_col = 'application_id',
```

```
                        amount_col = 'amount',
                        time_col = 'transaction_date',
                        ops = list(
                          avg_30_day_transaction_count_last_90_days = mean
                        ),
                        period = c(30, 3),
                        period_agg = length,
                        observation_window_start_col = 'obs_start',
                        scrape_date_col = 'scrape_date'
)

# Example 4: Aggregate transactions without time segmentation.
aggregate_applications(featForge_transactions,
                        id_col = 'application_id',
                        amount_col = 'amount',
                        ops = list(
                          total_transactions_counted = length,
                          total_outgoing_transactions_counted = function(x) sum(x < 0),
                          total_incoming_transactions_counted = function(x) sum(x > 0)
                        ),
                        period = 'all'
)
```

`aggregate_psd2_keyword_features` *Aggregate PSD2 Keyword Features at the Application Level with Time Window Filtering*

Description

This function extracts keyword features from a transaction descriptions column using the `extract_keyword_features` function and then aggregates these features at the application level using the `aggregate_applications` function. In addition, when the aggregation period is provided as a numeric vector (e.g., `c(30, 3)`), the function filters out transactions that fall outside the observation window defined as the period between `scrape_date - (period[1] * period[2])` and `scrape_date`. This prevents spending time processing keywords from transactions that would later be aggregated as zeros.

Usage

```
aggregate_psd2_keyword_features(
  data,
  id_col,
  description_col,
  amount_col = NULL,
  time_col = NULL,
  observation_window_start_col = NULL,
  scrape_date_col = NULL,
  ops = NULL,
```

```
  period = "all",
  separate_direction = if (!is.null(amount_col)) TRUE else FALSE,
  group_cols = NULL,
  min_freq = 1,
  use_matrix = TRUE,
  convert_to_df = TRUE,
  period_agg = sum,
  period_missing_inputs = 0
)
```

Arguments

data	A data frame containing transaction records.
id_col	A character string specifying the column name that identifies each application (e.g., "application_id").
description_col	A character string specifying the column name that contains the transaction descriptions. Note that this column may contain NA values.
amount_col	Optional. A character string specifying the column name that contains transaction amounts. If provided, the function aggregates a value for each keyword (default ops = list(amount = sum)). If omitted (NULL), the function aggregates counts of keyword occurrence (default ops = list(count = sum)).
time_col	Optional. A character string specifying the column name that contains the transaction date (or timestamp). When period is a numeric vector, this is required to filter the data by observation window.
observation_window_start_col	Optional. A character string indicating the column name with the observation window start date. If period is not "all" and is not numeric, this column is used in aggregate_applications.
scrape_date_col	Optional. A character string indicating the column name with the scrape date. If period is not "all" and is not numeric, this column is used in aggregate_applications.
ops	A named list of functions used to compute summary features on the aggregated values. If amount_col is provided and ops is NULL, the default is list(amount = sum). If amount_col is NULL and ops is NULL, the default is list(count = sum).
period	Either a character string or a numeric vector controlling time aggregation. The default is "all", meaning no time segmentation. If a numeric vector is provided (e.g., c(30, 3)), it defines a cycle length in days (first element) and a number of consecutive cycles (second element). In that case, only transactions with a transaction date between scrape_date - (period[1] * period[2]) and scrape_date are considered.
separate_direction	Logical. If TRUE (the default when amount_col is provided), a new column "direction" is added to automatically separate incoming and outgoing transactions based on the sign of the amount.

`group_cols` Optional. A character vector of additional grouping columns to use during aggregation. If `separate_direction` is `TRUE`, the `"direction"` grouping is added automatically.

`min_freq` Numeric. The minimum frequency a token must have to be included in the keyword extraction. Default is 1.

`use_matrix` Logical. Passed to `extract_keyword_features`; if `TRUE` (the default) a sparse matrix is used.

`convert_to_df` Logical. Passed to `extract_keyword_features`; if `TRUE` (the default) the sparse matrix is converted to a data.frame, facilitating binding with other data.

`period_agg` A function used to aggregate values within each period (see `aggregate_applications`). Default is `sum`.

`period_missing_inputs`
A numeric value to replace missing aggregated values. Default is `0`.

Details

The function supports two modes:

- If `amount_col` is not provided (i.e., `NULL`), the function aggregates keyword counts (i.e., the number of transactions in which a keyword appears) for each application.
- If `amount_col` is provided, then for each transaction the keyword indicator is multiplied by the transaction amount. In this mode, the default aggregation operation is to sum these values (using `ops = list(amount = sum)`), yielding the total amount associated with transactions that mention each keyword.

Additionally, if `amount_col` is provided and `separate_direction` is `TRUE` (the default), a new column named `"direction"` is created to separate incoming (`"in"`) and outgoing (`"out"`) transactions based on the sign of the amount. Any additional grouping columns can be provided via `group_cols`.

The function performs the following steps:

1. Basic input checks are performed to ensure the required columns exist.
2. The full list of application IDs is stored from the original data.
3. If `amount_col` is provided and `separate_direction` is `TRUE`, a `"direction"` column is added to label transactions as incoming (`"in"`) or outgoing (`"out"`) based on the sign of the amount.
4. When `period` is provided as a numeric vector, the function computes the observation window as `scrape_date - (period[1] * period[2])` to `scrape_date` and filters the dataset to include only transactions within this window. Transactions for applications with no records in the window will later be assigned zeros.
5. Keyword features are extracted from the `description_col` using `extract_keyword_features`. If an `amount_col` is provided, the binary indicators are weighted by the transaction amount.
6. The extracted keyword features are combined with the (possibly filtered) original data.
7. For each keyword, the function calls `aggregate_applications` to aggregate the feature by application. The aggregation is performed over time periods defined by `period` (if applicable) and, if requested, further split by direction.

8. Aggregated results for each keyword are merged by application identifier.
9. Finally, the aggregated results are merged with the full list of application IDs so that applications with no transactions in the observation window appear with zeros.

Value

A data frame with one row per application and aggregated keyword features.

Examples

```
# Example: Aggregate keyword features for PSD2 transactions.

data(featForge_transactions)

# In this example, the 'description' field is parsed for keywords.
# Since the 'amount' column is provided, each keyword indicator is
# weighted by the transaction amount, and transactions are
# automatically split into incoming and outgoing via the 'direction' column.
# Additionally, the period is specified as c(30, 1), meaning only
# transactions occurring within the last 30 days.
# (scrape_date - 30 to scrape_date) are considered.
result <- aggregate_psd2_keyword_features(
  data = featForge_transactions,
  id_col = "application_id",
  description_col = "description",
  amount_col = "amount",
  time_col = "transaction_date",
  scrape_date_col = "scrape_date",
  observation_window_start_col = "obs_start",
  period = c(30, 1),
  ops = list(amount = sum),
  min_freq = 1,
  use_matrix = TRUE,
  convert_to_df = TRUE
)

# The resulting data frame 'result' contains one
# row per application with aggregated keyword features.
# For example, if keywords "casino" and "utilities" were detected,
# aggregated columns might be named:
# "casino_amount_direction_in",
# "casino_amount_direction_out",
# "utilities_amount_direction_in", etc.
result
```

---

cyclical_cos *Convert a Cyclical Variable to Cosine Representation*

Description

This function transforms a cyclic variable (e.g., hour of the day, day of the week, month of the year) into its cosine representation. This transformation ensures that machine learning models respect the cyclical nature of such features.

Usage

```
cyclical_cos(x, period)
```

Arguments

x	A numeric vector representing a cyclic variable (e.g., hour, month, day).
period	A positive numeric scalar representing the period of the cycle. For example, use 24 for hours of the day, 7 for days of the week, and 12 for months.

Details

This function applies the transformation:

$$cos(2 * \pi * x/period)$$

This encoding ensures that the first and last values in the cycle are smoothly connected.

Value

A numeric vector representing the cosine-transformed values.

Examples

```
# Convert hours of the day to cosine encoding
hours <- 0:23
cyclical_cos(hours, 24)

# Convert months of the year to cosine encoding
months <- 1:12
cyclical_cos(months, 12)
```

---

cyclical_sin *Convert a Cyclical Variable to Sine Representation*

Description

This function transforms a cyclic variable (e.g., hour of the day, day of the week, month of the year) into its sine representation. This transformation ensures that machine learning models respect the cyclical nature of such features.

Usage

```
cyclical_sin(x, period)
```

Arguments

x	A numeric vector representing a cyclic variable (e.g., hour, month, day).
period	A positive numeric scalar representing the period of the cycle. For example, use 24 for hours of the day, 7 for days of the week, and 12 for months.

Details

This function applies the transformation:

$$sin(2 * \pi * x/period)$$

This encoding ensures that the first and last values in the cycle are smoothly connected.

Value

A numeric vector representing the sine-transformed values.

Examples

```
# Convert hours of the day to sine encoding
hours <- 0:23
cyclical_sin(hours, 24)

# Convert months of the year to sine encoding
months <- 1:12
cyclical_sin(months, 12)
```

---

extract_basic_description_features

Extract Basic Description Features

Description

This function processes a vector of text descriptions (such as transaction descriptions) and computes a set of basic text features. These features include counts of digits, special characters, punctuation, words, characters, unique characters, and letter cases, as well as word length statistics and the Shannon entropy of the text.

Usage

```
extract_basic_description_features(descriptions)
```

Arguments

descriptions	A character vector of text descriptions to be processed.

Details

The extracted features are:

has_digits A binary indicator (0/1) showing whether the description contains any digit.

n_digits The total count of digit characters in the description.

n_special The number of special characters (non-alphanumeric and non-whitespace) present.

n_punct The count of punctuation marks found in the description.

n_words The number of words in the description.

n_chars The total number of characters in the description.

n_unique_chars The count of unique characters in the description.

n_upper The count of uppercase letters in the description.

n_letters The total count of alphabetic characters (both uppercase and lowercase) in the description.

prop_caps The proportion of letters in the description that are uppercase.

n_whitespace The number of whitespace characters (spaces) in the description.

avg_word_length The average word length within the description.

min_word_length The length of the shortest word in the description.

max_word_length The length of the longest word in the description.

entropy The Shannon entropy of the description, indicating its character diversity.

The function uses vectorized string operations (e.g., `grepl`, `gregexpr`, and `nchar`) for efficiency, which makes it suitable for processing large datasets. The resulting numeric features can then be used directly for further statistical analysis or machine learning, or they can be aggregated to higher levels.

Value

A data frame where each row corresponds to an element in `descriptions` and each column represents a computed feature.

Examples

```
# Example 1: Extract features from a vector of sample descriptions.
descs <- c("KappaCredit#101",
           "Transferred funds for service fee 990",
           "Mighty remittance code 99816 casino")
extract_basic_description_features(descs)

# Example 2: Aggregate the maximum word length per application.
# Load the sample transactions data.
data(featForge_transactions)
```

```
# Combine the transactions data with extracted basic description features.
trans <- cbind(featForge_transactions,
               extract_basic_description_features(featForge_transactions$description))

# Aggregate the maximum word length on the application level.
aggregated <- aggregate_applications(
  trans,
  id_col = "application_id",
  amount_col = "max_word_length",
  ops = list(max_description_word_length = max),
  period = "all"
)

# Display the aggregated results.
aggregated
```

`extract_email_features` *Extract Email Features for Credit Scoring*

Description

This function processes a vector of email addresses to extract a comprehensive set of features that can be useful for credit scoring. In addition to parsing the email into its constituent parts (such as the username and domain), the function computes various character-level statistics (e.g., counts of digits, dots, uppercase letters) and string distance metrics between the email username and client name information. If provided, it also checks for the presence of date-of-birth components in the email username (in several flexible formats).

Usage

```
extract_email_features(
  emails,
  client_name = NULL,
  client_surname = NULL,
  date_of_birth = NULL,
  error_on_invalid = FALSE
)
```

Arguments

`emails`	A character vector of email addresses. Invalid email addresses are either replaced with `NA` (with a warning) or cause an error, depending on the value of `error_on_invalid`.
`client_name`	Optional. A character vector of client first names. When provided, its length must equal that of `emails`.

client_surname Optional. A character vector of client surnames. When provided, its length must equal that of emails.

date_of_birth Optional. A Date vector containing the client's dates of birth. When provided, its length must equal that of emails.

error_on_invalid
Logical. If TRUE, the function will throw an error when encountering an invalid email address. If FALSE (the default), invalid emails are replaced with NA and a warning is issued.

Details

The function is designed to support feature engineering for credit-scoring datasets. It not only extracts parts of an email address (such as the username and domain) but also computes detailed characteristics from the username, which may include embedded client information.

When client name information is provided, the function computes various string distance metrics (using the stringdist package) between the client name (and surname) and the email username. If stringdist is not installed, the function will issue a warning and assign NA to the distance-based features.

Value

A data.frame with the following columns:

email_domain The domain part of the email address (i.e., the substring after the '@').

email_major_domain The major domain extracted as the substring after the last dot in the domain (e.g., "com" in "gmail.com").

email_n_chars The number of characters in the email username (i.e., the part before the '@').

email_n_digits The number of digits found in the email username.

email_n_dots The number of dot ('.') characters in the email username.

email_n_caps The number of uppercase letters in the email username.

email_total_letters The total count of alphabetic characters (both uppercase and lowercase) in the email username.

email_prop_digits The proportion of digits in the email username (calculated as email_n_digits/email_n_chars).

email_max_consecutive_digits The maximum length of any sequence of consecutive digits in the email username.

email_name_in_email Logical. TRUE if the provided client name is found within the email username (case-insensitive), FALSE otherwise.

email_name_in_email_dist_lv The Levenshtein distance between the client name and the email username (if the stringdist package is available; otherwise NA).

email_name_in_email_dist_lcs The Longest Common Subsequence distance between the client name and the email username (if computed).

email_name_in_email_dist_cosine The cosine distance between the client name and the email username (if computed).

email_name_in_email_dist_jaccard The Jaccard distance between the client name and the email username (if computed).

`email_name_in_email_dist_jw` The Jaro-Winkler distance between the client name and the email username (if computed).

`email_name_in_email_dist_soundex` The Soundex distance between the client name and the email username (if computed).

`email_surname_in_email` Logical. `TRUE` if the provided client surname is found within the email username (case-insensitive), `FALSE` otherwise.

`email_surname_in_email_dist_lv` The Levenshtein distance between the client surname and the email username (if computed).

`email_surname_in_email_dist_lcs` The Longest Common Subsequence distance between the client surname and the email username (if computed).

`email_surname_in_email_dist_cosine` The cosine distance between the client surname and the email username (if computed).

`email_surname_in_email_dist_jaccard` The Jaccard distance between the client surname and the email username (if computed).

`email_surname_in_email_dist_jw` The Jaro-Winkler distance between the client surname and the email username (if computed).

`email_surname_in_email_dist_soundex` The Soundex distance between the client surname and the email username (if computed).

`email_fullname_in_email_dist_lv` The Levenshtein distance between the concatenated client name and surname and the email username (if computed).

`email_fullname_in_email_dist_lcs` The Longest Common Subsequence distance between the concatenated client name and surname and the email username (if computed).

`email_fullname_in_email_dist_cosine` The cosine distance between the concatenated client name and surname and the email username (if computed).

`email_fullname_in_email_dist_jaccard` The Jaccard distance between the concatenated client name and surname and the email username (if computed).

`email_fullname_in_email_dist_jw` The Jaro-Winkler distance between the concatenated client name and surname and the email username (if computed).

`email_fullname_in_email_dist_soundex` The Soundex distance between the concatenated client name and surname and the email username (if computed).

`email_has_full_year_of_birth` Logical. `TRUE` if the full 4-digit year (e.g., "1986") of the client's date of birth is present in the email username.

`email_has_last_two_digits_of_birth` Logical. `TRUE` if the last two digits of the client's birth year are present in the email username.

`email_has_full_dob_in_username` Logical. `TRUE` if the full date of birth (in one of the following formats: YYYYMMDD, YYYY.MM.DD, YYYY_MM_DD, or YYYY-MM-DD) is present in the email username.

`email_has_other_4digit_year` Logical. `TRUE` if a different 4-digit year (between 1920 and 2020) is found in the email username that does not match the client's own birth year.

See Also

`stringdist` for the calculation of string distances.

Examples

```
 # Load sample data included in the package
 data("featForge_sample_data")

# Extract features from the sample emails
 features <- extract_email_features(
   emails = featForge_sample_data$email,
   client_name = featForge_sample_data$client_name,
   client_surname = featForge_sample_data$client_surname,
   date_of_birth = featForge_sample_data$date_of_birth
 )

# Display the first few rows of the resulting feature set
head(features)
```

`extract_ip_features` *Extract IP Address Features*

Description

This function extracts a comprehensive set of features from a vector of IP address strings to support feature engineering in credit-scoring datasets. It processes both IPv4 and IPv6 addresses and returns a data frame with derived features. The features include IP version classification, octet-level breakdown for IPv4 addresses (with both string- and numeric-based octets), checks for leading zeros, a numeric conversion of the address, a basic approximation of IPv6 numeric values, pattern metrics such as a palindrome check and Shannon entropy, multicast status, and a Hilbert curve encoding for IPv4 addresses.

Usage

```
extract_ip_features(ip_addresses, error_on_invalid = FALSE)
```

Arguments

`ip_addresses`	A character vector of IP address strings.
`error_on_invalid`	Logical flag indicating how to handle invalid IP addresses. If `TRUE`, the function throws an error upon encountering any invalid IP address; if `FALSE` (the default), invalid IP addresses are replaced with `NA` and a warning is issued.

Details

The function follows these steps:

- **Validation:** Each IP address is checked against regular expressions for both IPv4 and IPv6. If an IP does not match either pattern, it is deemed invalid. Depending on the value of `error_on_invalid`, invalid entries are either replaced with `NA` (with a warning) or cause an error.

- **IP Version Identification:** The function determines whether an IP address is IPv4 or IPv6.
- **IPv4 Feature Extraction:**
  - The IPv4 addresses are split into four octets.
  - For each octet, both the raw (string) and numeric representations are extracted.
  - The presence of leading zeros is checked for each octet, and the total count of octets with leading zeros is computed.
  - The full IPv4 address is converted to a 32-bit numeric value.
  - Hilbert curve encoding is applied to the numeric value, yielding two dimensions that can be used as features in modeling.
- **IPv6 Feature Extraction:** For IPv6 addresses, an approximate numeric conversion is performed to allow for coarse interval analysis.
- **Pattern Metrics:** Independent of IP version, the function computes:
  - A palindrome check on the entire IP string.
  - The Shannon entropy of the IP string to capture the diversity of characters.

Value

A data frame with the following columns:

`ip_version` A character vector indicating the IP version; either `"IPv4"` or `"IPv6"`. Invalid addresses are set to `NA`.

`ip_v4_octet1` The numeric conversion of the first octet of an IPv4 address as extracted from the IP string.

`ip_v4_octet2` The numeric conversion of the second octet of an IPv4 address.

`ip_v4_octet3` The numeric conversion of the third octet of an IPv4 address.

`ip_v4_octet4` The numeric conversion of the fourth octet of an IPv4 address.

`ip_v4_octet1_has_leading_zero` An integer flag indicating whether the first octet of an IPv4 address includes a leading zero.

`ip_v4_octet2_has_leading_zero` An integer flag indicating whether the second octet includes a leading zero.

`ip_v4_octet3_has_leading_zero` An integer flag indicating whether the third octet includes a leading zero.

`ip_v4_octet4_has_leading_zero` An integer flag indicating whether the fourth octet includes a leading zero.

`ip_leading_zero_count` An integer count of how many octets in an IPv4 address contain leading zeros.

`ip_v4_numeric_vector` The 32-bit integer representation of an IPv4 address, computed as $(A * 256^3) + (B * 256^2) + (C * 256) + D$.

`ip_v6_numeric_approx_vector` An approximate numeric conversion of an IPv6 address. This value is computed from the eight hextets and is intended for interval comparisons only; precision may be lost for large values (above 2^53).

`ip_is_palindrome` An integer value indicating whether the entire IP address string is a palindrome (i.e., it reads the same forwards and backwards).

`ip_entropy` A numeric value representing the Shannon entropy of the IP address string, computed over the distribution of its characters. Higher entropy values indicate a more varied (less repetitive) pattern.

Examples

```
# Load the package's sample dataset
data(featForge_sample_data)

# Extract IP features and combine them with the original IP column
result <- cbind(
  data.frame(ip = featForge_sample_data$ip),
  extract_ip_features(featForge_sample_data$ip)
)
print(result)
```

`extract_keyword_features` *Extract Keyword Features from Text Descriptions*

Description

This function processes a vector of text descriptions (e.g., transaction descriptions) and extracts binary keyword features. Each unique word (token) that meets or exceeds the specified minimum frequency is turned into a column that indicates its presence (1) or absence (0) in each description. For efficiency, the function can create a sparse matrix using the Matrix package. Additionally, the user can choose to convert the output to a standard data.frame.

Usage

```
extract_keyword_features(
  descriptions,
  min_freq = 1,
  use_matrix = TRUE,
  convert_to_df = TRUE
)
```

Arguments

`descriptions`	A character vector of text descriptions.
`min_freq`	A numeric value specifying the minimum frequency a token must have to be included in the output. Default is 1.
`use_matrix`	Logical. If `TRUE` (the default), a sparse matrix is created using the Matrix package.
`convert_to_df`	Logical. If `TRUE` (the default) and `use_matrix` is `TRUE`, the resulting sparse matrix is converted to a dense data.frame. This is useful for binding with other data.frames, but may be memory-intensive if many unique tokens exist.

Details

The function performs the following steps:

1. Converts the input text to lowercase and removes digits.
2. Splits the text on any non-letter characters to extract tokens.
3. Constructs a vocabulary from all tokens that appear at least `min_freq` times.
4. Depending on the `use_matrix` flag, builds either a sparse matrix or a dense matrix of binary indicators.
5. If `use_matrix` is `TRUE` and `convert_to_df` is also `TRUE`, the sparse matrix is converted to a data.frame.

Value

Either a data.frame or a sparse matrix (if `convert_to_df` is `FALSE`) with one row per description and one column per keyword. Each cell is binary (1 if the keyword is present, 0 otherwise).

Warnings

- If a very large number of descriptions (e.g., more than 100,000) is provided with `use_matrix = FALSE`, a warning is issued because using a dense matrix may exhaust memory.
- If `use_matrix = TRUE`, `convert_to_df = TRUE`, and `min_freq` is set to 1 (i.e., no filtering), a warning is issued because converting a very large sparse matrix to a dense data.frame may result in an enormous object.

Examples

```
# Load the sample transactions data.
data(featForge_transactions)

# Combine the transactions data with extracted keyword features.
trans <- cbind(featForge_transactions,
               extract_keyword_features(featForge_transactions$description))
head(trans) # we see that there have been added categories named "casino" and "utilities".

# Aggregate the number of transactions on the application level
# by summing the binary keyword indicators for the 'casino' and
# 'utilities' columns, using a 30-day period.
aggregate_applications(
  trans,
  id_col = "application_id",
  amount_col = "amount",
  group_cols = c("casino", "utilities"),
  time_col = "transaction_date",
  scrape_date_col = "scrape_date",
  ops = list(times_different_transactions_in_last_30_days = sum),
  period_agg = length,
  period = c(30, 1)
)
```

`extract_timestamp_features` *Extract Timestamp Features*

Description

This function extracts various features from application timestamps and, if provided, client dates of birth. It supports both `POSIXct` and `Date` objects for `timestamps`. If the timestamps are given as `Date` objects, note that features requiring intra-day granularity (e.g., `timestamp_hour`) will not be created, and some cyclic features may be less precise.

Usage

```
extract_timestamp_features(
  timestamps,
  date_of_birth = NULL,
  error_on_invalid = FALSE
)
```

Arguments

`timestamps`	A vector of timestamps, either as `POSIXct` or `Date` objects.
`date_of_birth`	An optional vector of client dates of birth. If provided, it must have the same length as `timestamps`, enabling computation of age and birthday-related features.
`error_on_invalid`	Logical flag specifying whether to throw an error (`TRUE`) or a warning (`FALSE`, default) when missing or invalid timestamp values are detected.

Details

The function returns a data frame containing the following variables:

timestamp_month Numeric. Month extracted from the timestamp (1 to 12).

timestamp_month_sine Numeric. Sine transformation of the month (using period = 12).

timestamp_month_cosine Numeric. Cosine transformation of the month (using period = 12).

timestamp_day_of_month Numeric. Day of the month extracted from the timestamp (1 to 31).

timestamp_day_of_month_sine Numeric. Sine transformation of the day of the month (using period = 31).

timestamp_day_of_month_cosine Numeric. Cosine transformation of the day of the month (using period = 31).

timestamp_week_of_year Numeric. ISO week number extracted from the timestamp (typically 1 to 52, but may be 53 in some years).

timestamp_week_of_year_sine Numeric. Sine transformation of the week of the year (using period = 52).

timestamp_week_of_year_cosine Numeric. Cosine transformation of the week of the year (using period = 52).

timestamp_day_of_week Numeric. Day of the week extracted from the timestamp (1 for Monday through 7 for Sunday).

timestamp_day_of_week_sine Numeric. Sine transformation of the day of the week (using period = 7).

timestamp_day_of_week_cosine Numeric. Cosine transformation of the day of the week (using period = 7).

timestamp_hour Numeric. Hour of the day (0 to 23). This is only available if `timestamps` are of class `POSIXct`.

timestamp_hour_sine Numeric. Sine transformation of the hour (using period = 24).

timestamp_hour_cosine Numeric. Cosine transformation of the hour (using period = 24).

client_age_at_application Numeric. Client's age at the time of application, calculated in years (real number).

days_to_birthday Numeric. Number of days until the client's next birthday.

days_to_birthday_cosine Numeric. Cosine transformation of the days to birthday (using period = 365).

The function first validates the inputs and then extracts the following features:

- Extracts date-based features such as year, month, day of month, ISO week of the year, and day of the week.
- If timestamps are of class `POSIXct`, it also extracts the hour of the day.
- Applies cyclic transformations (using sine and cosine functions) to the month, day of month, week of year, day of week, and hour variables so that their cyclical nature is maintained in machine learning models.
- If `date_of_birth` is provided, computes the client's age at the time of application and the number of days until their next birthday, along with a cosine transformation for the latter.

Value

A data frame with the extracted timestamp features and birthday-related features (if `date_of_birth` is provided).

Examples

```
# Load sample data
data(featForge_sample_data)

# Generate features and combine with the original dataset
result <- cbind(
  data.frame(
    application_created_at = featForge_sample_data$application_created_at,
    client_date_of_birth = featForge_sample_data$date_of_birth
  ),
  extract_timestamp_features(
    featForge_sample_data$application_created_at,
```

```
    featForge_sample_data$date_of_birth
  )
)

head(result)
```

`featForge_sample_data` *Sample Data for Package Testing and Demonstration*

Description

A dataset containing auto-generated values by ChatGPT o3-mini-high for package testing and demonstration purposes. This example dataset consists of 30 rows and 7 variables that illustrate the structure of a typical application record.

Usage

```
data(featForge_sample_data)
```

Format

A data frame with 30 rows and 7 variables.

Details

The dataset includes the following variables:

application_id A unique numeric identifier for each application.

application_created_at The date and time when the application was created.

client_name The first name of the client.

client_surname The last name of the client.

date_of_birth The client's date of birth.

email The client's email address.

ip The IP address associated with the client.

This dataset was automatically generated for the purpose of testing and demonstrating the package functionality. The values (e.g., dates, emails, and IP addresses) are illustrative and do not represent actual user data.

Examples

```
# Load the dataset
data(featForge_sample_data)

# Display the first few rows of the dataset
head(featForge_sample_data)
```

`featForge_transactions` *Transactions Data for Package Testing and Demonstration*

Description

A dataset containing auto-generated banking transactions for package testing and demonstration purposes. This example dataset consists of over 200 rows and 5 variables that illustrate the structure of a typical transaction record associated with an application.

Usage

```
data(featForge_transactions)
```

Format

A data frame with over 200 rows and 5 variables.

Details

The dataset includes the following variables:

application_id A numeric identifier linking the transaction to the corresponding application.

scrape_date The end date of the transactions observation window. Assumed to be on the same date with the application's creation date.

obs_start The start date of the transactions observation window. Assumed to be 180 days apart with the scrape_date.

transaction_date The date and time when the transaction occurred. The date is always on or before the associated scrape_date (application's creation date) and on or after the obs_start date.

amount The monetary amount of the transaction. Negative values indicate outgoing transactions, while positive values indicate incoming transactions.

description A detailed description of the transaction. Descriptions vary in style and content, including combinations of generated institution names, random numbers, special characters, creative phrases, and full English sentences. For transactions with a category of "gambling", the description always includes the word "casino".

category The transaction category, such as groceries, salary, gambling, utilities, travel, entertainment, rent, or shopping.

This dataset was automatically generated for the purpose of testing and demonstrating the package functionality. The transactions simulate realistic banking activity linked to application records. The transaction dates are generated in relation to each application's creation date, ensuring logical consistency in the timeline, while the descriptions have been enriched to include a wide variety of content, with special attention given to transactions in the gambling category.

Examples

```
# Load the transactions dataset
data(featForge_transactions)

# Display the first few rows of the dataset
head(featForge_transactions)
```

Index