Package 'PVplr'

February 14, 2023

Type Package

Title Performance Loss Rate Analysis Pipeline

Version 0.1.2

Description The pipeline contained in this package provides tools used in the Solar Durability and Lifetime Extension Center (SDLE) for the analysis of Performance Loss Rates (PLR) in real world photovoltaic systems. Functions included allow for data cleaning, feature correction, power predictive modeling, PLR determination, and uncertainty bootstrapping through various methods <doi:10.1109/PVSC40753.2019.8980928>.
The vignette ``Pipeline Walkthrough'' gives an explicit run through of typical package usage.
This material is based upon work supported by the U.S Department of Energy's Office of Energy Efficiency and Renewable Energy (EERE) under Solar Energy Technologies Office (SETO) Agreement Number DE-EE-0008172.
This work made use of the High Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University.

Depends R (>= 2.10)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr (>= 0.7.8), magrittr (>= 1.5), broom (>= 0.5.1), ggplot2 (>= 3.1.0), stlplus (>= 0.5.1), cluster (>= 2.0.7-1), purrr (>= 0.3.3), tidyr (>= 1.1.1), minpack.lm (>= 1.2-1), rlang (>= 0.3.1), segmented, forecast, scales, zoo

RoxygenNote 7.2.3

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Alan Curran [aut] (<https://orcid.org/0000-0002-4505-8359>),
Tyler Burleyson [aut] (<https://orcid.org/0000-0002-6356-5354>),
William Oltjen [aut] (<https://orcid.org/0000-0003-0380-1033>),

Sascha Lindig [aut] (<https://orcid.org/0000-0001-5421-8265>),
David Moser [aut] (<https://orcid.org/0000-0002-4895-8862>),
Roger French [aut, cre] (<https://orcid.org/0000-0002-6162-0532>),
Solar Durability and Lifetime Extension research center [cph, fnd]

Maintainer Roger French <roger.french@case.edu>

Repository CRAN

Date/Publication 2023-02-14 18:00:02 UTC

R topics documented:

`all_na` *function to test if an entire column is NA*

Description

This function tests for completely NA columns

Usage

```
all_na(x)
```

Arguments

`x`	any column in a dataframe

Value

Returns boolean TRUE if column is all NA, FALSE if not

Examples

```
test <- all_na(c(NA, "a", NA))
```

`anomalies` *Fixes the anomlies*

Description

This function gets the data and finds the anomlies in weekends and weekdays and gives a dataframe with anomalies and anomaly columns

Usage

```
anomalies(df)
```

Arguments

`df`	structured dataframe

Value

df with two columns of cleaned_energy and anom_flag

Author(s)

Arash Khalilnejad

`anomaly_detector` *detects rhw anomalies and returns a dataframw with cleaned and anom_flag column*

Description

detects rhw anomalies and returns a dataframw with cleaned and anom_flag column

Usage

```
anomaly_detector(df, batch_days = 90)
```

Arguments

`df`	the strucutred data
`batch_days`	the batch of data that the anomaly detection is applied. Since time series decomposition is used, one seasonality will be applied for whole data which is inefficeint, if NA, will pass whole

Value

data with anomalies

Author(s)

Arash Khalilnejad

`data_quality_check` *checks the quality of the data after and before cleaning*

Description

calculates the percentage of anomalies, missings + zeros, gaps, and length of the data and reports the quality of data before and after cleaning.

Usage

```
data_quality_check(
  energy_data,
  col = "elec_cons",
  id = "pv_df",
  batch_days = 90
)
```

Arguments

`energy_data`	structured energy dataframe
`col`	Input column
`id`	PV system ID
`batch_days`	the batch of data that the anomaly detection is applied. Since time series decomposition is used, one seasonality will be applied for whole data which is inefficient, if NA, will pass whole

Details

The quality grading criteria is as following: anomalies A: less than 10 missing percentage: A: less than 10 largest gap: A: less than 120 hours, B: 120 to 164 hours, C: 164 to 240 hours D: more than 240 hours length P: more than 2 years, F: less than 2 years

Value

a table with grading of the quality after and before cleaning

Author(s)

Arash Khalilnejad

`data_structure` *Reads jci files gotten in budget period 2*

Description

Reads the jci file and modifies the timestamp intevals and based on location modifies the timezone using googleapi and then generates the useful columns

Usage

```
data_structure(df, col = "elec_cons", timestamp_col = "timestamp")
```

Arguments

`df`	dataframe containing at least the timestamp column and the variable to be plotted with the heatmap
`col`	the character name of the column to be ploted
`timestamp_col`	the character name of the timestamp column which i is the number of file in the list

Value

a dataframe with fixed timestamps and useful cooumns

Author(s)

Arash

`day_time_start_end` *finds median start and end time of PV operation*

Description

finds median start and end time of PV operation

Usage

```
day_time_start_end(df)
```

Arguments

`df`	with num_time Column

Value

dataframe with start and end time

Author(s)

Arash Khalilnejad

`df_With_on_time` *data with PV on time flag.*

Description

returns dataframe of PV with approximate operating period, baed on median of start and end time.

Usage

```
df_With_on_time(df)
```

Arguments

`df`	df with num_time

Value

input data with one more column of on_time

Author(s)

Arash Khalilnejad

`grade_pv` *returns quality information of time series data of PV*

Description

returns quality information of time series data of PV

Usage

```
grade_pv(
  df,
  col = "poay",
  id = "pv_id",
  timestamp_col = "tmst",
  timestamp_format = "%Y-%m-%d %H:%M:%S",
  batch_days = 90
)
```

Arguments

`df`	the PV time series data. It can be the direct output of read.csv(file_name, stringsAsFactors = F)
`col`	column of the grading, default 'poay'
`id`	The name of the pv data
`timestamp_col`	the character name of the timestamp column
`timestamp_format`	the POSIXct format of the timestamp if conversion is needed
`batch_days`	the batch of data that the anomaly detection is applied. Since time series decomposition is used, one seasonality will be applied for whole data which is inefficeint, if NA, will pass whole

Author(s)

Arash Khalilnejad

`Int` *Largest Intervals*

Description

Largest Intervals

Usage

```
Int(df)
```

Arguments

`df`	Dataframe

Value

Intervals

Author(s)

Arash Khalilnejad

ip_num_time — *Numerical time interim predictor.*

Description

Convert the hour and minute component of each timestamp to a numerical representation.

Usage

```
ip_num_time(data, ts_col = "timestamp")
```

Arguments

`data`	A dataframe with a timestamp column.
`ts_col`	The timestamp column name in `data`. Default value is 'timestamp'.

Value

`data` with a num_time column added.

Author(s)

Arash Khalilnejad

lin_inter_hrly_to_fifteen — *Linearly interpolate hourly data to 15 min data.*

Description

Many weather data sets are hourly and we need values for every 15 minutes.

Usage

```
lin_inter_hrly_to_fifteen(data, data_ts)
```

Arguments

`data`	A data frame with hourly data.
`data_ts`	The column name for the `data` timestamp.

Details

Any value that can not be linearly interpolated such as a string will remain the same.

Value

The resulting fifteen minute data frame.

Author(s)

Arash Khalilnejad

`lin_inter_missing_energy`
Linearly interpolate missing energy values.

Description

If there exist lest than four missing values, represented by NA values, fill with linearly interpolated values.

Usage

```
lin_inter_missing_energy(data, threshold = 4)
```

Arguments

`data`	A data frame with an 'elec_cons' column.
`threshold`	The maximum number of consective values that may be filled with interpolated values. By default four.

Value

The data frame with 'missing values' filled in.

Examples

```
## Not run:
lin_inter_missing_energy(data)

## End(Not run)
```

mbm_resample — *Dataframe resample function*

Description

This function resamples data from a given dataframe. Dataframe must have columns created through plr_cleaning to denote time segments

Usage

```
mbm_resample(df, fraction, by)
```

Arguments

df	dataframe
fraction	fraction of data to resample from dataframe
by	timescale over which to resample, day, week, or month

Value

Returns randomly resampled dataframe

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

dfc_resampled <- mbm_resample(test_dfc, fraction = 0.65, by = "week")
```

nc — *function to convert to character then numeric*

Description

The function is a shorthand for converting factors to numeric

Usage

```
nc(x)
```

Arguments

`x`	any factor to convert to numeric

Value

Returns supplied parameter as numeric

Examples

```
num <- nc(test_df$power)
```

`num_test` *function to test is the values in a column should be numeric*

Description

This function tests a column to see if it should be numeric

Usage

```
num_test(col)
```

Arguments

`col`	any column in a dataframe

Value

Returns boolean TRUE if column should be numeric, FALSE if not

Examples

```
test <- num_test(test_df$power)
```

parallel_cluster_export *Export variables to a cluster.*

Description

Ghost cluster export call to make sure testCoverage's trace function and environment are available.

Usage

```
parallel_cluster_export(cluster, varlist, envir = .GlobalEnv)
```

Arguments

cluster	Cluster
varlist	Character vector of names of objects to export.
envir	Environment from which t export variables

plr_6k_model *6k Method for PLR Determination*

Description

This function groups data by the specified time interval and performs a linear regression using the formula: power_var ~ irrad_var/istc * (nameplate_power + a*log(irrad_var/istc) + b*log(irrad_var/istc)^2 + c*(temp_var - tref) + d*(temp_var - tref)*log(irrad_var/istc) + e*(temp_var - tref)*log(irrad_var/istc)^2 + f*(temp_var - tref)^2). Predicted values of irradiance, temperature, and wind speed (if applicable) are added for reference. These values are the lowest daily high irradiance reading (over 300W/m^2), the average temperature over all data, and the average wind speed over all data.

Usage

```
plr_6k_model(
  df,
  var_list,
  nameplate_power,
  by = "month",
  data_cutoff = 30,
  predict_data = NULL
)
```

Arguments

df	A dataframe containing pv data.
var_list	A list of the dataframe's standard variable names, obtained from the output of plr_variable_check.
nameplate_power	The rated power capability of the system, in watts.
by	String, either "day", "week", or "month". The time periods over which to group data for regression.
data_cutoff	The number of data points needed to keep a value in the final table. Regressions over less than this number and their data will be discarded.
predict_data	optional; Dataframe; If you have preferred estimations of irradiance, temperature, and wind speed, include them here to skip automatic generation. Format: Irradiance, Temperature, Wind (optional).

Value

Returns dataframe of results per passed time scale from 6K modeling

---

plr_bootstrap_output *Bootstrap: Resampling from individual Models*

Description

This function determines uncertainty of a PLR measurement by sampling results from invididual models. Specify the model you would like to find the uncertainty of, and the function will put the dataframe through the selected model and return the uncertainties of the model's results.

Usage

```
plr_bootstrap_output(
  df,
  var_list,
  model,
  by = "month",
  fraction = 0.65,
  n = 1000,
  predict_data = NULL,
  np = NA,
  power_var = "power_var",
  time_var = "time_var",
  ref_irrad = 900,
  irrad_range = 10
)
```

Arguments

df	A dataframe containing pv data.
var_list	A list of the dataframe's standard variable names, obtained from the plr_variable_check output.
model	The model you would like to calculate the uncertainty of. Use "xbx", "xbx+utc", "pvusa", or "6k".
by	String indicating time step count per year for the regression. Use "day", "month", or "year". See plr_weighted_regression.
fraction	The size of each sample relative to the total dataset.
n	Number of samples to take.
predict_data	passed to predict_data in model call. See plr_xbx_model for example.
np	The system's reported name plate power. See plr_6k_model.
power_var	The name of the power variable after being put through a Performance Loss Rate (PLR) determining test. Typically "power_var".
time_var	The name of the time variable after being put through a PLR determining test. Typically "time_var".
ref_irrad	The irradiance value at which to calculate the universal temperature coefficient. Since irradiance is a much stronger influencer on power generation than temperature, it is important to specify a small range of irradiance data from which to estimate the effect of temperature.
irrad_range	The range of the subset used to calculate the universal temperature coefficient. See above.

Value

Returns PLR value and uncertainty calculated with bootstrap of data from power correction models

Examples

```
# build var_list

var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

xbx_mbm_plr_output_uncertainty <- plr_bootstrap_output(test_dfc, var_list,
                                                       model = "xbx", fraction = 0.65,
                                                       n = 10, power_var = 'power_var',
                                                  time_var = 'time_var', ref_irrad = 900,
                                                       irrad_range = 10, by = "month",
                                                       np = NA, pred = NULL)
```

plr_bootstrap_output_from_results *Bootstrap: Resample from individual Models*

Description

The function samples and bootstraps data that has already been put through a power predictive model. The PLR and Uncertainty are returned in a dataframe.

Usage

```
plr_bootstrap_output_from_results(
  data,
  power_var,
  time_var,
  weight_var,
  by = "month",
  model,
  fraction = 0.65,
  n = 1000
)
```

Arguments

data	Result of modeling data with a PLR determining model, i.e. plr_xbx_model, plr_6k_model, etc.
power_var	Variable name of power in the dataframe. Typically power_var
time_var	Variable name of time in the dataframe. Typically time_var
weight_var	Variable name of weightings in the dataframe. Typically sigma
by	String, either "day", "month", or "year". Time over which to perform `plr_yoy_regression` and `plr_weighted_regression`.
model	The name of the model the data has been put through. This option is only included for the user's benefit in keeping bootstrap outputs consistent.
fraction	The fractional size of the data to be sampled each time.
n	The number of resamples to take.

Value

Returns PLR value and uncertainty calculated with bootstrap of data going into power correction models

Examples

```
# build var_list


var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

xbx_mbm_plr_result_uncertainty <- plr_bootstrap_output_from_results(test_xbx_wbw_res,
                                                                    power_var = 'power_var',
                                                                      time_var = 'time_var',
                                                                       weight_var = 'sigma',
                                                                    by = "month", model = 'xbx',
                                                                    fraction = 0.65, n = 10)
```

---

plr_bootstrap_uncertainty

Bootstrap: Resampling data going into each Model

Description

This function determines the uncertainty of a PLR measurement through resampling data for each model, prior to putting the data through the model.

Usage

```
plr_bootstrap_uncertainty(
  df,
  n,
  fraction = 0.65,
  var_list,
  model,
  by = "month",
  power_var = "power_var",
  time_var = "time_var",
  data_cutoff = 100,
  np = NA,
  pred = NULL
)
```

Arguments

- `df` — A dataframe containing pv data.
- `n` — (numeric) Number of samples to take. The higher the n value, the longer it takes to complete, but the results become more accurate as well.
- `fraction` — The fraction of data that constitutes a resample for the bootstrap.
- `var_list` — A list of variables obtained through `plr_variable_check`.
- `model` — the String name of the model to bootstrap. Select from:
  - "xbx" (`plr_xbx_model`),
  - "correction" (`plr_xbx_utc_model`),
  - "pvusa" (`plr_pvusa_model`),
  - or "6k" (`plr_6k_model`).
- `by` — String, either "day", "week", or "month". Time over which to perform `plr_yoy_regression`.
- `power_var` — Variable name of power in the dataframe. This must be the variable's name after being put through your selected model. Typically power_var
- `time_var` — Variable name of time in the dataframe. This must be the variable's name after being put through your selected model. Typically time_var
- `data_cutoff` — The number of data points needed to keep a value in the final table. Regressions over less than this number and their data will be discarded.
- `np` — The system's reported name plate power. See `plr_6k_model`.
- `pred` — passed to predict_data in model call. See `plr_xbx_model` for an example.

Value

Returns PLR value and uncertainty calculated with bootstrap of data going into power correction models

Examples

```
# build var_list


var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

xbx_mbm_plr_uncertainty <- plr_bootstrap_uncertainty(test_dfc, n = 2,
                                                     fraction = 0.65, by = 'month',
                                          power_var = 'power_var', time_var = 'time_var',
                                                     var_list = var_list, model = "xbx",
                                                     data_cutoff = 10, np = NA,
                                                     pred = NULL)
```

plr_build_var_list — *Build a Custom Variable List*

Description

The default var_list generator, plr_variable_check, assumes data comes from SDLE's sources. If you are using this package with your own data, the format may not line up appropriately. Use this function to create a variable list to be passed to other functions so they can keep track of what column names mean.

Usage

```
plr_build_var_list(time_var, power_var, irrad_var, temp_var, wind_var)
```

Arguments

`time_var`	The variable representing time. Typically, a timestamp.
`power_var`	The variable representing time. Typically, in watts.
`irrad_var`	The variable representing irradiance. Typically, either poa or ghi irradiance.
`temp_var`	The variable representing temperature. Package functions assume Celcius.
`wind_var`	optional; The variable representing wind speed.

Value

Returns dataframe of variable names for the given photovoltaic data for use with later functions

Examples

```
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
```

plr_cleaning — *Basic Data Cleaning*

Description

Removes entries with irradiance and power readings outside cutoffs, fixes timestamps to your specified format, and converts columns to numeric when appropriate - see `plr_convert_columns`. Also, adds columns for days/weeks/years of operation that are used by other functions.

Usage

```
plr_cleaning(
  df,
  var_list,
  irrad_thresh = 100,
  low_power_thresh = 0.05,
  high_power_cutoff = NA,
  tmst_format = "%Y-%m-%d %H:%M:%S"
)
```

Arguments

df	A dataframe containing pv data.
var_list	A list of the dataframe's standard variable names, obtained from the output of plr_variable_check.
irrad_thresh	The lowest meaningful irradiance value. Values below are filtered.
low_power_thresh	The lowest meaningful power output. Values below are filtered.
high_power_cutoff	The highest meaningful power output. Values above are filtered.
tmst_format	The desired timestamp format.

Value

Returns dataframe with rows filtered out based on passed cleaning parameters

Examples

```
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)

test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)
```

---

plr_convert_columns *Fix Column Typings*

Description

Converts appropriate columns to numeric without specifying the name of the column. All columns from hbase are read as factors. Columns are tested to see if they should be numeric by forcing conversion to numeric. Columns that subsequently contain NA's are not numeric; if not, they are set to numeric.

Usage

```
plr_convert_columns(df)
```

Arguments

df	A dataframe containing pv data.

Value

Returns original dataframe with columns corrected to proper classes

Examples

```
df <- PVplr::plr_convert_columns(test_df)
```

---

plr_decomposition *Decompose Seasonality from Data*

Description

Decomposes seasonality from a dataframe that has already passed through a PLR Determination test, e.g. plr_xbx_model. This method has the option of creating plot and data files.

Usage

```
plr_decomposition(
  data,
  freq,
  power_var,
  time_var,
  plot = FALSE,
  plot_file = NULL,
  title = NULL,
  data_file = NULL
)
```

Arguments

data	a dataframe containing PV data that has undergone a power predictive model, e.g. plr_xbx_model.
freq	the frequency of seasonality. This is typically 4 but depends on the location of the system.
power_var	name of the power variable, e.g. iacp
time_var	name of the time variable, e.g. tvar
plot	boolean indicating if you wish to save a plot.

plot_file	location to save the plot, if the plot param is given TRUE.
title	the title of the plot created if the plot param is given TRUE.
data_file	location to save data. Currently non-functional.

Value

Dataframe containing decomposed time series features

Examples

```
#' # build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)
# Perform power modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

test_xbx_wbw_decomp <- plr_decomposition(test_xbx_wbw_res, freq = 4,
                                         power_var = 'power_var', time_var = 'time_var',
                                         plot = FALSE, plot_file = NULL, title = NULL,
                                         data_file = NULL)
```

---

plr_kmeans_test *Statistical k-means Test*

Description

The method builds linear models by day, identifies outliers, and performs 2-means clustering by slopes. If the lower identified cluster is significantly less than the higher mean, and constitutes less than 25% of the data, it is identified as soiled and returned. Otherwise, the outlier points are identified as soiled and returned.

Usage

```
plr_kmeans_test(
  df,
  var_list,
  mean_ratio = 0.7,
  plot = FALSE,
  file_path,
  file_name,
  set_cutoff = FALSE
)
```

Arguments

df	A df containing pv data. Should be 'cleaned' by plr_cleaning.
var_list	A list of the dataframe's standard variable names, obtained from the output of plr_variable_check.
mean_ratio	This scales the higher identified cluster's mean for comparison. Higher values will be more likely to identify the second mean as soiled, and vice versa. Values should range from 0 to 1.
plot	optional; Boolean; whether to return the box plot generated by the method to identify outliers.
file_path	optional; location to store the boxplot if plot is set TRUE. Note this is not necessary if you select to plot - only if you wish to save it.
file_name	optional; name of file to save boxplot if plot is set to TRUE.
set_cutoff	Defaults to FALSE; pass a numeric value to cut off all slopes less than the cutoff value. This bypasses entirely the outlier and clustering calculuations to remove slope values you believe to be soiled.

Value

The method returns a dataframe containing the values that should be removed. If you want to discard them, try using dplyr::filter().

---

plr_pvheatmap *Title Heatmap generation for PV data*

Description

Title Heatmap generation for PV data

Usage

```
plr_pvheatmap(
  df,
  col,
  timestamp_col,
  timestamp_format = "%Y-%m-%d %H:%M:%S",
  upper_threshold = 1,
  lower_threshold = 0,
  font_size = 12
)
```

Arguments

- `df` — dataframe containing at least the timestamp column and the variable to be plotted with the heatmap
- `col` — the character name of the column to be ploted
- `timestamp_col` — the character name of the timestamp column
- `timestamp_format` — the POSIXct format of the timestamp if conversion is needed
- `upper_threshold` — the fraction of upper data to include, 1 removes no data, 0.9 remove the top 1 percent etc.
- `lower_threshold` — the fraction of lower data to remove, 0 removes no data, 0.01 remove the bottom 1 percent etc.
- `font_size` — font size of the output plot

Value

returns a ggplot object heatmap of the specified column

Examples

```
# build heatmap
heat <- plr_pvheatmap(test_df, col = "g_poa", timestamp_col = "timestamp",
                       upper_threshold = 0.99, lower_threshold = 0)
# display heatmap
plot(heat)
```

`plr_pvusa_model` *PVUSA Method for PLR Determination*

Description

This function groups data by the specified time interval and performs a linear regression using the formula: $P = G_{POA} * (\beta_0 + \beta_1 G + \beta_2 T_{amb} + \beta_3 W)$. Predicted values of irradiance, temperature, and wind speed (if applicable) are added for reference. These values are the lowest daily high irradiance reading (over 300), the average temperature over all data, and the average wind speed over all data.

Usage

```
plr_pvusa_model(
  df,
  var_list,
  by = "month",
  data_cutoff = 30,
  predict_data = NULL
)
```

Arguments

`df`	A dataframe containing pv data.
`var_list`	A list of the dataframe's standard variable names, obtained from the output of `plr_variable_check`.
`by`	String, either "day", "week", or "month". The time periods over which to group data for regression.
`data_cutoff`	The number of data points needed to keep a value in the final table. Regressions over less than this number and their data will be discarded.
`predict_data`	optional; Dataframe; If you have preferred estimations of irradiance, temperature, and wind speed, include them here to skip automatic generation. Format: Irradiance, Temperature, Wind (optional).

Value

Returns dataframe of results per passed time scale from PVUSA modeling

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_pvusa_model(test_dfc, var_list, by = "week",
                                    data_cutoff = 30, predict_data = NULL)
```

`plr_remove_outliers` *Filter outliers from Power Predicted Data*

Description

This function is used to remove outliers (if desired) after putting data through a power predictive model, e.g. `plr_xbx_model`.

Usage

```
plr_remove_outliers(data)
```

Arguments

`data`	A resulting dataframe from a power predictive model.

Value

Returns dataframe with outliers flagged by other functions removed

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

# Remove outliers from the modeled data
test_xbx_wbw_res_no_outliers <- plr_remove_outliers(test_xbx_wbw_res)
```

---

plr_saturation_removal *Removing Saturated Data*

Description

Tests for readings which may indicate saturation of the system. Removes values above the power saturation limit (calculated by multiplying sat_limit and power_thresh).

Usage

```
plr_saturation_removal(df, var_list, sat_limit, power_thresh = 0.99)
```

Arguments

df	A dataframe containing pv data.
var_list	A list of the dataframe's standard variable names, obtained from the output of plr_variable_check.
sat_limit	An upper limit on power saturation. This is multiplied by the power threshold, and power values above this point are filtered from the dataframe. The value depends on the system's inverter.
power_thresh	An upper limit on power.

Value

Returns passed data frame with rows removed which contain power values above the specified threshold

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

test_dfc_removed_saturation <- plr_saturation_removal(test_dfc, var_list,
                                                      sat_limit = 3000, power_thresh = 0.99)
```

---

plr_seg_extract — *Segmented linear PLR extraction function*

Description

Segmented linear PLR extraction function

Usage

```
plr_seg_extract(
  df,
  per_year,
  psi = NA,
  n_breakpoints,
  power_var,
  time_var,
  return_model = FALSE
)
```

Arguments

df	data frame of corrected power measurements, typically the output of a weather correction model
per_year	number of data point defining one seasonal year (365 for days, 52 for weeks etc.)
psi	vector of 1 or more breakpoint estimates for the model. If not given will evenly space breakpoints across time series

n_breakpoints	number of desired breakpoints. Determines number of linear models
power_var	character name of the power variable
time_var	character name of the time variable
return_model	logical to return model object. If FALSE returns PLR results from model

Value

if return_model is FALSE it returns PLR results from model, otherwise returns segmented linear model object

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

#' # Perform power modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

decomp <- plr_decomposition(test_xbx_wbw_res, freq = 4,
                                         power_var = 'power_var', time_var = 'time_var',
                                         plot = FALSE, plot_file = NULL, title = NULL,
                                         data_file = NULL)

# evaluate segmented PLR results
seg_plr_result <- PVplr::plr_seg_extract(df = decomp, per_year = 365,
                                         n_breakpoints = 1, power_var = "trend",
                                         time_var = "age")

# return segmented model instead of PLR result
model <- PVplr::plr_seg_extract(df = decomp, per_year = 365, n_breakpoints = 1,
                             power_var = "trend", time_var = "age", return_model = TRUE)

# predict data along time-series with piecewise model for plotting
pred <- data.frame(age = seq(1, max(decomp$age, na.rm = TRUE), length.out = 10000))
pred$seg <- predict(model, newdata = pred)
```

---

plr_var *PLR linear model uncertainty*

Description

This function returns the standard deviation of a PLR calculated from a linear model

Usage

```
plr_var(mod, per_year)
```

Arguments

`mod`	linear model
`per_year`	number of data points in a given year baesd on which time scale was selected

Value

Returns standard deviation of PLR value

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

# obain standard deviation from model
mod <- lm(power_var ~ time_var, data = test_xbx_wbw_res)
plr_sd <- plr_var(mod, per_year = 52)
```

`plr_variable_check` *Define Standard Variable Names*

Description

The method determines the variable names used by the input dataframe. It looks for the following labels:

- power_var <- iacp; if not, sets to idcp
- time_var <- tmst; if not ,sets to tutc
- irrad_var <- poay; if not, sets to ghir

- temp_var <- temp; if not, sets to modt
- wind_var <- wspa; if applicable, else NULL

This function assumes data is in a standard HBase format. If you are using other data (as you most likely are) you should use the companion function, plr_build_var_list.

Usage

```
plr_variable_check(df)
```

Arguments

df	A dataframe containing pv data.

Value

Returns a dataframe containing standard variable names (no data). It will not include windspeed if the variable was not already included. This is frequently an input of other functions.

Examples

```
var_list <- plr_variable_check(test_df)
```

---

plr_weighted_regression *Weighted Regression*

Description

Automatically calculates Performance Loss Rate (PLR) using weighted linear regression. Note that it needs data from a power predictive model.

Usage

```
plr_weighted_regression(
  data,
  power_var,
  time_var,
  model,
  per_year = 12,
  weight_var = NA
)
```

Arguments

`data`	The result of a power predictive model
`power_var`	String name of the variable used as power
`time_var`	String name of the variable used as time
`model`	String name of the model that the data was passed through
`per_year`	the time step count per year based on the model - 12 for month-by-month, 52 for week-by-week, and 365 for day-by-day
`weight_var`	Used to weight regression, typically sigma.

Value

Returns PLR value and error evaluated with linear regression

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

# Calculate Performance Loss Rate
xbx_wbw_plr <- plr_weighted_regression(test_xbx_wbw_res,
                                       power_var = 'power_var',
                                       time_var = 'time_var',
                                       model = "xbx",
                                       per_year = 52,
                                       weight_var = 'sigma')
```

`plr_xbx_model` *XbX Method for PLR Determination*

Description

This function groups data by the specified time interval and performs a linear regression using the formula: $P_{pred.} = \beta_0 + \beta_1 G + \beta_2 T + \epsilon$. This is the simplest of the PLR determining methods. Predicted values of irradiance, temperature, and wind speed (if applicable) are added to the output for reference. These values are the lowest daily high irradiance reading (over 300), the average temperature over all data, and the average wind speed over all data. Outliers are detected and labeled in a column as TRUE or FALSE.

Usage

```
plr_xbx_model(
  df,
  var_list,
  by = "month",
  data_cutoff = 30,
  predict_data = NULL
)
```

Arguments

df	A dataframe containing pv data.
var_list	A list of the dataframe's standard variable names, obtained from the plr_variable_check output.
by	String, either "day", "week", or "month". The time periods over which to group data for regression.
data_cutoff	The number of data points needed to keep a value in the final table. Regressions over less than this number and their data will be discarded.
predict_data	optional; Dataframe; If you have preferred estimations of irradiance, temperature, and wind speed, include them here to skip automatic generation. Format: Irradiance, Temperature, Wind (optional).

Value

Returns dataframe of results per passed time scale from XbX modeling

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)
```

plr_xbx_utc_model *UTC Method for PLR Determination*

Description

This function groups data by the specified time interval and performs a linear regression using the formula: power_corr ~ irrad_var - 1. Predicted values of irradiance, temperature, and wind speed (if applicable) are added for reference. The function uses a universal temperature correction, rather than the monthly regression correction done in other PLR determining methods.

Usage

```
plr_xbx_utc_model(
  df,
  var_list,
  by = "month",
  data_cutoff = 30,
  predict_data = NULL,
  ref_irrad = 900,
  irrad_range = 10
)
```

Arguments

df	A dataframe containing pv data.
var_list	A list of the dataframe's standard variable names, obtained from the output of plr_variable_check.
by	String, either "day", "week", or "month". The time periods over which to group data for regression.
data_cutoff	The number of data points needed to keep a value in the final table. Regressions over less than this number and their data will be discarded.
predict_data	optional; Dataframe; If you have preferred estimations of irradiance, temperature, and wind speed, include them here to skip automatic generation. Format: Irradiance, Temperature, Wind (optional).
ref_irrad	The irradiance value at which to calculate the universal temperature coefficient. Since irradiance is a much stronger influencer on power generation than temperature, it is important to specify a small range of irradiance data from which to estimate the effect of temperature.
irrad_range	The range of the subset used to calculate the universal temperature coefficient. See above.

Value

Returns dataframe of results per passed time scale from XbX with universal temperature correction modeling

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_utc_model(test_dfc, var_list, by = "week",
                                      data_cutoff = 30, predict_data = NULL,
                                      ref_irrad = 900, irrad_range = 10)
```

`plr_yoy_regression` *Year-on-Year Regression*

Description

Automatically calculates Performance Loss Rate (PLR) using year on year regression. Note that it needs data from a power predictive model.

Usage

```
plr_yoy_regression(
  data,
  power_var,
  time_var,
  model,
  per_year = 12,
  return_PLR = TRUE
)
```

Arguments

`data`	Result of a power predictive model
`power_var`	String name of the variable used as power
`time_var`	String name of the variable used as time
`model`	String name of the model the data was passed through
`per_year`	Time step count per year based on model. Typically 12 for MbM, 365 for DbD.
`return_PLR`	boolean; option to return PLR value, rather than the raw regression data.

Value

Returns PLR value and error evaluated with YoY regression, if return_PLR is false it will return the individual YoY calculations

Examples

```
# build var_list
var_list <- plr_build_var_list(time_var = "timestamp",
                               power_var = "power",
                               irrad_var = "g_poa",
                               temp_var = "mod_temp",
                               wind_var = NA)
# Clean Data
test_dfc <- plr_cleaning(test_df, var_list, irrad_thresh = 100,
                         low_power_thresh = 0.01, high_power_cutoff = NA)

# Perform the power predictive modeling step
test_xbx_wbw_res <- plr_xbx_model(test_dfc, var_list, by = "week",
                                  data_cutoff = 30, predict_data = NULL)

# Calculate Performance Loss Rate
xbx_wbw_plr <- plr_yoy_regression(test_xbx_wbw_res,
                                       power_var = 'power_var',
                                       time_var = 'time_var',
                                       model = "xbx",
                                       per_year = 52,
                                       return_PLR = TRUE)
```

---

spline_timestamp_sync *Spline columns to match timestamps.*

Description

Often timestamps of two data frames will be mismatched. To produced matching timestamps, columns that may be splined will be and then corresponding values at the 'correct' timestamp are used.

Usage

```
spline_timestamp_sync(
  data,
  data_ts = "timestamp",
  merge_data,
  merge_ts = "timestamp"
)
```

Arguments

`data`	A data frame with a correct timestamp column.
`data_ts`	The column name for the `data` timestamp. Defaults to 'timestamp'
`merge_data`	A data frame that will be linearly interpolated and merged with `data`.
`merge_ts`	The column name for the `merge_data` timestamp. Defaults to 'timestamp'.

Details

Any value that can not be linearly interpolated such as a string will remain the same.

Value

The resulting merged data frame.

Author(s)

Arash Khalilnejad

`test_df` *DOE RTC Sample PV System Data*

Description

A dataset containing a small, randomly taken sample of PV data from SDLE's data collection. It is included for the purposes of unit tests and vignettes, serving as an example of how the package's functions work.

Usage

```
test_df
```

Format

A .csv file that can be read as a dataframe. 16265 rows and 22 variables.

time_frequency — *Determines the minutes between data points in a time-series*

Description

Determines the minutes between data points in a time-series

Usage

```
time_frequency(data)
```

Arguments

`data`	A time-series dataframe containing a column named 'timestamp'.

Value

a numeric value of the minutes between data points

Author(s)

Arash Khalilnejad

ts_inflate — *Inflate a time series data set.*

Description

Shifts known values to the nearest equidistant timestamp and fills in any missing timestamps with NA values. An additional binary column named `<column to impute>_imp` is added where 1 represents an unknown value and zero represents a known value.

Usage

```
ts_inflate(data, ts_col, col_to_imp, dt)
```

Arguments

`data`	A data frame containing columns `ts_col` and `col_to_imp`.
`ts_col`	The name of the timestamp column.
`col_to_imp`	The name of the column to impute.
`dt`	The expected time between consecutive timestamps, in minutes.

Index