Package 'vetiver'

October 12, 2022

Title Version, Share, Deploy, and Monitor Models

Version 0.1.8

Description The goal of 'vetiver' is to provide fluent tooling to version, share, deploy, and monitor a trained model. Functions handle both recording and checking the model's input data prototype, and predicting from a remote API endpoint. The 'vetiver' package is extensible, with generics that can support many kinds of models.

License MIT + file LICENSE

URL https://vetiver.rstudio.com, https://rstudio.github.io/vetiver-r/, https://github.com/rstudio/vetiver-r/

BugReports https://github.com/rstudio/vetiver-r/issues

Depends R (>= 3.4)

Imports bundle, butcher, cli, fs, generics, glue, hardhat, lifecycle, magrittr (>= 2.0.3), pins (>= 1.0.0), plumber (>= 1.0.0), purrr, rapidoc, readr (>= 1.4.0), renv, rlang (>= 1.0.0), tibble, vctrs, withr

Suggests callr, caret, covr, curl, dplyr, flexdashboard, ggplot2, httpuv, httr, jsonlite, knitr, LiblineaR, mgcv, mlr3, mlr3data, mlr3learners, modeldata, parsnip, pingr, plotly, ranger, recipes, rmarkdown, rpart, rsconnect, slider (>= 0.2.2), stacks, testthat (>= 3.0.0), tidyselect, vdiffr, workflows, xgboost, yardstick

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate, rstudio/connectapi

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author Julia Silge [cre, aut] (<https://orcid.org/0000-0002-3671-836X>), RStudio [cph, fnd]

Maintainer Julia Silge <julia.silge@rstudio.com>

Repository CRAN

Date/Publication 2022-09-29 18:10:03 UTC

R topics documented:

`api_spec` *Update the OpenAPI specification using model metadata*

Description

Update the OpenAPI specification using model metadata

Usage

```
api_spec(spec, vetiver_model, path, all_docs = TRUE)

glue_spec_summary(ptype, return_type)

## Default S3 method:
glue_spec_summary(ptype, return_type = NULL)
```

```
## S3 method for class 'data.frame'
glue_spec_summary(ptype, return_type = "predictions")

## S3 method for class 'array'
glue_spec_summary(ptype, return_type = "predictions")
```

Arguments

spec	An OpenAPI Specification formatted list object
vetiver_model	A deployable vetiver_model() object
path	The endpoint path
all_docs	Should the interactive visual API documentation be created for *all* POST endpoints in the router pr? This defaults to TRUE, and assumes that all POST endpoints use the vetiver_model$ptype input data prototype.
ptype	An input data prototype from a model
return_type	Character string to describe what endpoint returns, such as "predictions"

Value

api_spec() returns the updated OpenAPI Specification object. This function uses glue_spec_summary() internally, which returns a glue character string.

Examples

```
library(plumber)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")

glue_spec_summary(v$ptype)

modify_spec <- function(spec) api_spec(spec, v, "/predict")
pr() %>% pr_set_api_spec(api = modify_spec)
```

---

attach_pkgs *Fully attach or load packages for making model predictions*

Description

These are developer-facing functions, useful for supporting new model types. Some models require one or more R packages to be fully attached to make predictions, and some require only that the namespace of one or more R packages is loaded.

Usage

```
attach_pkgs(pkgs)

load_pkgs(pkgs)
```

Arguments

pkgs	A character vector of package names to load or fully attach.

Details

These two functions will attempt either to:

- fully attach or
- load

the namespace of the pkgs vector of package names, preserving the current random seed.

To learn more about load vs. attach, read the "Dependencies" chapter of *R Packages*. For deploying a model, it is likely safer to fully attach needed packages but that comes with the risk of naming conflicts between packages.

Value

An invisible TRUE.

Examples

```
## succeed
load_pkgs(c("knitr", "readr"))
attach_pkgs(c("knitr", "readr"))

## fail
try(load_pkgs(c("bloopy", "readr")))
try(attach_pkgs(c("bloopy", "readr")))
```

---

augment.vetiver_endpoint

Post new data to a deployed model API endpoint and augment with predictions

---

Description

Post new data to a deployed model API endpoint and augment with predictions

Usage

```
## S3 method for class 'vetiver_endpoint'
augment(x, new_data, ...)
```

Arguments

x	A model API endpoint object created with vetiver_endpoint().
new_data	New data for making predictions, such as a data frame.
...	Extra arguments passed to httr::POST()

Value

The new_data with added prediction column(s).

See Also

predict.vetiver_endpoint()

Examples

```
if (FALSE) {
endpoint <- vetiver_endpoint("http://127.0.0.1:8088/predict")
augment(endpoint, mtcars[4:7, -1])
}
```

---

handler_startup.train *Model handler functions for API endpoint*

Description

These are developer-facing functions, useful for supporting new model types. Each model supported by vetiver_model() uses two handler functions in vetiver_api():

- The handler_startup function executes when the API starts. Use this function for tasks like loading packages. A model can use the default method here, which is NULL (to do nothing at startup).
- The handler_predict function executes at each API call. Use this function for calling predict() and any other tasks that must be executed at each API call.

Usage

```
## S3 method for class 'train'
handler_startup(vetiver_model)

## S3 method for class 'train'
handler_predict(vetiver_model, ...)

## S3 method for class 'gam'
handler_startup(vetiver_model)
```

```
## S3 method for class 'gam'
handler_predict(vetiver_model, ...)

## S3 method for class 'glm'
handler_predict(vetiver_model, ...)

handler_startup(vetiver_model)

## Default S3 method:
handler_startup(vetiver_model)

handler_predict(vetiver_model, ...)

## Default S3 method:
handler_predict(vetiver_model, ...)

## S3 method for class 'lm'
handler_predict(vetiver_model, ...)

## S3 method for class 'Learner'
handler_startup(vetiver_model)

## S3 method for class 'Learner'
handler_predict(vetiver_model, ...)

## S3 method for class 'ranger'
handler_startup(vetiver_model)

## S3 method for class 'ranger'
handler_predict(vetiver_model, ...)

## S3 method for class 'model_stack'
handler_startup(vetiver_model)

## S3 method for class 'model_stack'
handler_predict(vetiver_model, ...)

## S3 method for class 'workflow'
handler_startup(vetiver_model)

## S3 method for class 'workflow'
handler_predict(vetiver_model, ...)

## S3 method for class 'xgb.Booster'
handler_startup(vetiver_model)

## S3 method for class 'xgb.Booster'
handler_predict(vetiver_model, ...)
```

Arguments

vetiver_model	A deployable vetiver_model() object
...	Other arguments passed to predict(), such as prediction type

Details

These are two generics that use the class of vetiver_model$model for dispatch.

Value

A handler_startup function should return invisibly, while a handler_predict function should return a function with the signature function(req). The request body (req$body) consists of the new data at prediction time; this function should return predictions either as a tibble or as a list coercable to a tibble via tibble::as_tibble().

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
handler_startup(v)
handler_predict(v)
```

---

map_request_body *Identify data types for each column in an input data prototype*

Description

The OpenAPI specification of a Plumber API created via plumber::pr() can be modified via plumber::pr_set_api_spec(), and this helper function will identify data types of predictors and create a list to use in this specification. These are *not* R data types, but instead basic JSON data types. For example, factors in R will be documented as strings in the OpenAPI specification.

Usage

```
map_request_body(ptype)
```

Arguments

ptype	An input data prototype from a model

Details

This is a developer-facing function, useful for supporting new model types. It is called by api_spec().

Value

A list to be used within `plumber::pr_set_api_spec()`

Examples

```
map_request_body(vctrs::vec_slice(chickwts, 0))
```

`predict.vetiver_endpoint` *Post new data to a deployed model API endpoint and return predictions*

Description

Post new data to a deployed model API endpoint and return predictions

Usage

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

Arguments

`object`	A model API endpoint object created with `vetiver_endpoint()`.
`new_data`	New data for making predictions, such as a data frame.
`...`	Extra arguments passed to `httr::POST()`

Value

A tibble of model predictions with as many rows as in `new_data`.

See Also

`augment.vetiver_endpoint()`

Examples

```
if (FALSE) {
endpoint <- vetiver_endpoint("http://127.0.0.1:8088/predict")
predict(endpoint, mtcars[4:7, -1])
}
```

vetiver_api *Create a Plumber API to predict with a deployable* `vetiver_model()` *object*

Description

Use `vetiver_api()` to add a POST endpoint for predictions from a trained `vetiver_model()` to a Plumber router.

Usage

```
vetiver_api(
  pr,
  vetiver_model,
  path = "/predict",
  debug = is_interactive(),
  ...
)

vetiver_pr_post(
  pr,
  vetiver_model,
  path = "/predict",
  debug = is_interactive(),
  ...,
  check_ptype = TRUE
)

vetiver_pr_docs(pr, vetiver_model, path = "/predict", all_docs = TRUE)
```

Arguments

`pr`	A Plumber router, such as from `plumber::pr()`.
`vetiver_model`	A deployable `vetiver_model()` object
`path`	The endpoint path
`debug`	`TRUE` provides more insight into your API errors.
`...`	Other arguments passed to `predict()`, such as prediction `type`
`check_ptype`	Should the `ptype` stored in `vetiver_model` (used for visual API documentation) also be used to check new data at prediction time? Defaults to `TRUE`.
`all_docs`	Should the interactive visual API documentation be created for *all* POST endpoints in the router `pr`? This defaults to `TRUE`, and assumes that all POST endpoints use the `vetiver_model$ptype` input data prototype.

Details

You can first store and version your `vetiver_model()` with `vetiver_pin_write()`, and then create an API endpoint with `vetiver_api()`.

Setting `debug = TRUE` may expose any sensitive data from your model in API errors.

Two GET endpoints will also be added to the router `pr`, depending on the characteristics of the model object: a `/pin-url` endpoint to return the URL of the pinned model and a `/ping` endpoint for the API health.

The function `vetiver_api()` uses:

- `vetiver_pr_post()` for endpoint definition and
- `vetiver_pr_docs()` to create visual API documentation

These modular functions are available for more advanced use cases.

Value

A Plumber router with the prediction endpoint added.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")

library(plumber)
pr() %>% vetiver_api(v)
## is the same as:
pr() %>% vetiver_pr_post(v) %>% vetiver_pr_docs(v)
## for either, next, pipe to `pr_run()`
```

---

vetiver_compute_metrics

Aggregate model metrics over time for monitoring

Description

These three functions can be used for model monitoring (such as in a monitoring dashboard):

- `vetiver_compute_metrics()` computes metrics (such as accuracy for a classification model or RMSE for a regression model) at a chosen time aggregation `period`
- `vetiver_pin_metrics()` updates an existing pin storing model metrics over time
- `vetiver_plot_metrics()` creates a plot of metrics over time

Usage

```
vetiver_compute_metrics(
  data,
  date_var,
  period,
  truth,
  estimate,
  ...,
  metric_set = yardstick::metrics,
  every = 1L,
  origin = NULL,
  before = 0L,
  after = 0L,
  complete = FALSE
)
```

Arguments

data	A data.frame containing the columns specified by truth, estimate, and ....
date_var	The column in data containing dates or date-times for monitoring, to be aggregated with .period
period	[character(1)] A string defining the period to group by. Valid inputs can be roughly broken into: • "year", "quarter", "month", "week", "day" • "hour", "minute", "second", "millisecond" • "yweek", "mweek" • "yday", "mday"
truth	The column identifier for the true results (that is numeric or factor). This should be an unquoted column name although this argument is passed by expression and support quasiquotation (you can unquote column names).
estimate	The column identifier for the predicted results (that is also numeric or factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name.
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected. Otherwise, there should be as many columns as factor levels of truth.
metric_set	A yardstick::metric_set() function for computing metrics. Defaults to yardstick::metrics().
every	[positive integer(1)] The number of periods to group together. For example, if the period was set to "year" with an every value of 2, then the years 1970 and 1971 would be placed in the same group.

origin [Date(1) / POSIXct(1) / POSIXlt(1) / NULL]
: The reference date time value. The default when left as NULL is the epoch time of 1970-01-01 00:00:00, *in the time zone of the index*.
: This is generally used to define the anchor time to count from, which is relevant when the every value is > 1.

before, after [integer(1) / Inf]
: The number of values before or after the current element to include in the sliding window. Set to Inf to select all elements before or after the current element. Negative values are allowed, which allows you to "look forward" from the current element if used as the .before value, or "look backwards" if used as .after.

complete [logical(1)]
: Should the function be evaluated on complete windows only? If FALSE, the default, then partial computations will be allowed.

Details

For arguments used more than once in your monitoring dashboard, such as date_var, consider using R Markdown parameters to reduce repetition and/or errors.

Value

A dataframe of metrics.

See Also

vetiver_pin_metrics(), vetiver_plot_metrics()

Examples

```
library(dplyr)
library(parsnip)
data(Chicago, package = "modeldata")
Chicago <- Chicago %>% select(ridership, date, all_of(stations))
training_data <- Chicago %>% filter(date < "2009-01-01")
testing_data <- Chicago %>% filter(date >= "2009-01-01", date < "2011-01-01")
monitoring <- Chicago %>% filter(date >= "2011-01-01", date < "2012-12-31")
lm_fit <- linear_reg() %>% fit(ridership ~ ., data = training_data)

library(pins)
b <- board_temp()

original_metrics <-
    augment(lm_fit, new_data = testing_data) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)

new_metrics <-
    augment(lm_fit, new_data = monitoring) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
```

`vetiver_create_description.train` *Model constructor methods*

Description

These are developer-facing functions, useful for supporting new model types. Each model supported by `vetiver_model()` uses up to four methods when the deployable object is created:

- The `vetiver_create_description()` function generates a helpful description of the model based on its characteristics. This method is required.
- The `vetiver_create_meta()` function creates the correct `vetiver_meta()` for the model. This is especially helpful for specifying which packages are needed for prediction. A model can use the default method here, which is to have no special metadata.
- The `vetiver_ptype()` function finds an input data prototype from the training data (a zero-row slice) to use for checking at prediction time. This method is required.
- The `vetiver_prepare_model()` function executes last. Use this function for tasks like checking if the model is trained and reducing the size of the model via `butcher::butcher()`. A model can use the default method here, which is to return the model without changes.

Usage

```
## S3 method for class 'train'
vetiver_create_description(model)

## S3 method for class 'train'
vetiver_prepare_model(model)

## S3 method for class 'gam'
vetiver_create_description(model)

## S3 method for class 'gam'
vetiver_prepare_model(model)

## S3 method for class 'glm'
vetiver_create_description(model)

## S3 method for class 'glm'
vetiver_prepare_model(model)

## S3 method for class 'lm'
vetiver_create_description(model)

## S3 method for class 'lm'
vetiver_prepare_model(model)
```

```
## S3 method for class 'Learner'
vetiver_create_description(model)

## S3 method for class 'Learner'
vetiver_prepare_model(model)

vetiver_create_description(model)

## Default S3 method:
vetiver_create_description(model)

vetiver_prepare_model(model)

## Default S3 method:
vetiver_prepare_model(model)

## S3 method for class 'ranger'
vetiver_create_description(model)

## S3 method for class 'ranger'
vetiver_prepare_model(model)

## S3 method for class 'model_stack'
vetiver_create_description(model)

## S3 method for class 'model_stack'
vetiver_prepare_model(model)

## S3 method for class 'workflow'
vetiver_create_description(model)

## S3 method for class 'workflow'
vetiver_prepare_model(model)

## S3 method for class 'xgb.Booster'
vetiver_create_description(model)

## S3 method for class 'xgb.Booster'
vetiver_prepare_model(model)
```

Arguments

model	A trained model, such as an `lm()` model or a tidymodels `workflows::workflow()`.

Details

These are four generics that use the class of `model` for dispatch.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
vetiver_create_description(cars_lm)
vetiver_prepare_model(cars_lm)
```

`vetiver_create_meta.train` *Metadata constructors for* `vetiver_model()` *object*

Description

These are developer-facing functions, useful for supporting new model types. The metadata stored in a `vetiver_model()` object has four elements:

- `$user`, the metadata supplied by the user
- `$version`, the version of the pin (which can be `NULL` before pinning)
- `$url`, the URL where the pin is located, if any
- `$required_pkgs`, a character string of R packages required for prediction

Usage

```
## S3 method for class 'train'
vetiver_create_meta(model, metadata)

## S3 method for class 'gam'
vetiver_create_meta(model, metadata)

vetiver_meta(user = list(), version = NULL, url = NULL, required_pkgs = NULL)

vetiver_create_meta(model, metadata)

## Default S3 method:
vetiver_create_meta(model, metadata)

## S3 method for class 'Learner'
vetiver_create_meta(model, metadata)

## S3 method for class 'ranger'
vetiver_create_meta(model, metadata)

## S3 method for class 'model_stack'
vetiver_create_meta(model, metadata)

## S3 method for class 'workflow'
vetiver_create_meta(model, metadata)
```

```
## S3 method for class 'xgb.Booster'
vetiver_create_meta(model, metadata)
```

Arguments

model	A trained model, such as an lm() model or a tidymodels workflows::workflow().
metadata	A list containing additional metadata to store with the pin. When retrieving the pin, this will be stored in the user key, to avoid potential clashes with the metadata that pins itself uses.
user	Metadata supplied by the user
version	Version of the pin
url	URL for the pin, if any
required_pkgs	Character string of R packages required for prediction

Value

The vetiver_meta() constructor returns a list. The vetiver_create_meta function returns a vetiver_meta() list.

Examples

```
vetiver_meta()

cars_lm <- lm(mpg ~ ., data = mtcars)
vetiver_create_meta(cars_lm, list())
```

---

vetiver_create_rsconnect_bundle

Create an RStudio Connect bundle for a vetiver model API

Description

Use vetiver_create_rsconnect_bundle() to create an RStudio Connect model API bundle for a vetiver_model() that has been versioned and stored via vetiver_pin_write().

Usage

```
vetiver_create_rsconnect_bundle(
  board,
  name,
  version = NULL,
  predict_args = list(),
  filename = fs::file_temp(pattern = "bundle", ext = ".tar.gz")
)
```

Arguments

board	A pin board, created by board_folder(), board_rsconnect(), board_url() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.
predict_args	A list of optional arguments passed to vetiver_api() such as the endpoint path or prediction type.
filename	The path for the model API bundle to be created (can be used as the argument to connectapi::bundle_path())

Details

This function creates a deployable bundle. See RStudio Connect docs for how to deploy this bundle, as well as the connectapi R package for how to integrate with Connect's API from R.

The two functions vetiver_create_rsconnect_bundle() and vetiver_deploy_rsconnect() are alternatives to each other, providing different strategies for deploying a vetiver model API to RStudio Connect.

Value

The location of the model API bundle filename, invisibly.

See Also

vetiver_write_plumber(), vetiver_deploy_rsconnect()

Examples

```
library(pins)
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

## when you pin to RStudio Connect, your pin name will be typically be like:
## "user.name/cars_linear"
vetiver_create_rsconnect_bundle(
    b,
    "cars_linear",
    predict_args = list(debug = TRUE)
)
```

`vetiver_dashboard` *R Markdown format for model monitoring dashboards*

Description

R Markdown format for model monitoring dashboards

Usage

```
vetiver_dashboard(pins, display_pins = TRUE, ...)

get_vetiver_dashboard_pins()

pin_example_kc_housing_model(board = pins::board_local(), name = "seattle_rf")
```

Arguments

`pins`	A list containing `board`, `name`, and `version`, as in `pins::pin_read()`
`display_pins`	Should the dashboard display a link to the pin(s)? Defaults to `TRUE`, but only creates a link if the pin contains a URL in its metadata.
`...`	Arguments passed to `flexdashboard::flex_dashboard()`
`board`	A pin board, created by `board_folder()`, `board_rsconnect()`, `board_url()` or another `board_` function.
`name`	Pin name.

Details

The `vetiver_dashboard()` function is a specialized type of **flexdashboard**. See the flexdashboard website for additional documentation: https://pkgs.rstudio.com/flexdashboard/

Before knitting the example `vetiver_dashboard()` template, execute the helper function `pin_example_kc_housing_model` to set up demonstration model and metrics pins needed for the monitoring demo. This function will:

- fit an example model to training data
- pin the vetiver model to your own `pins::board_local()`
- compute metrics from testing data
- pin these metrics to the same local board

These are the steps you need to complete before setting up monitoring your real model.

vetiver_deploy_rsconnect *Deploy a vetiver model API to RStudio Connect*

Description

Use `vetiver_deploy_rsconnect()` to deploy a `vetiver_model()` that has been versioned and stored via `vetiver_pin_write()` as a Plumber API on RStudio Connect.

Usage

```
vetiver_deploy_rsconnect(
  board,
  name,
  version = NULL,
  predict_args = list(),
  appTitle = glue::glue("{name} model API"),
  ...
)
```

Arguments

`board`	A pin board, created by `board_folder()`, `board_rsconnect()`, `board_url()` or another `board_` function.
`name`	Pin name.
`version`	Retrieve a specific version of a pin. Use `pin_versions()` to find out which versions are available and when they were created.
`predict_args`	A list of optional arguments passed to `vetiver_api()` such as the endpoint `path` or prediction `type`.
`appTitle`	The API title on RStudio Connect. Use the default based on `name`, or pass in your own title.
`...`	Other arguments passed to `rsconnect::deployApp()` such as `account` or `launch.browser`.

Details

The two functions `vetiver_deploy_rsconnect()` and `vetiver_create_rsconnect_bundle()` are alternatives to each other, providing different strategies for deploying a vetiver model API to RStudio Connect.

Value

The deployment success (`TRUE` or `FALSE`), invisibly.

See Also

`vetiver_write_plumber()`, `vetiver_create_rsconnect_bundle()`

Examples

```
library(pins)
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

if (FALSE) {
## pass args for predicting:
vetiver_deploy_rsconnect(
    b,
    "user.name/cars_linear",
    predict_args = list(debug = TRUE)
)

## specify an account name through `...`:
vetiver_deploy_rsconnect(
    b,
    "user.name/cars_linear",
    account = "user.name"
)
}
```

---

vetiver_endpoint *Create a model API endpoint object for prediction*

Description

This function creates a model API endpoint for prediction from a URL. No HTTP calls are made until you actually `predict()` with your endpoint.

Usage

```
vetiver_endpoint(url)
```

Arguments

`url`	An API endpoint URL

Value

A new `vetiver_endpoint` object

Examples

```
vetiver_endpoint("https://colorado.rstudio.com/rsc/seattle-housing/predict")
```

vetiver_model *Create a vetiver object for deployment of a trained model*

Description

A `vetiver_model()` object collects the information needed to store, version, and deploy a trained model. Once your `vetiver_model()` object has been created, you can:

- store and version it as a pin with `vetiver_pin_write()`
- create an API endpoint for it with `vetiver_api()`

Usage

```
vetiver_model(
  model,
  model_name,
  ...,
  description = NULL,
  metadata = list(),
  save_ptype = TRUE,
  versioned = NULL
)

new_vetiver_model(model, model_name, description, metadata, ptype, versioned)
```

Arguments

`model`	A trained model, such as an `lm()` model or a tidymodels `workflows::workflow()`.
`model_name`	Model name or ID.
`...`	Other method-specific arguments passed to `vetiver_ptype()` to compute an input data prototype, such as `ptype_data` (a sample of training features).
`description`	A detailed description of the model. If omitted, a brief description of the model will be generated.
`metadata`	A list containing additional metadata to store with the pin. When retrieving the pin, this will be stored in the `user` key, to avoid potential clashes with the metadata that pins itself uses.
`save_ptype`	Should an input data prototype be stored with the model? The options are `TRUE` (the default, which stores a zero-row slice of the training data), `FALSE` (no input data prototype for visual documentation or checking), or a dataframe to be used for both checking at prediction time *and* examples in API visual documentation.
`versioned`	Should the model object be versioned when stored with `vetiver_pin_write()`? The default, `NULL`, will use the default for the `board` where you store the model.
`ptype`	An input data prototype. If `NULL`, there is no checking of new data at prediction time.

Details

You can provide your own data to `save_ptype` to use as examples in the visual documentation created by `vetiver_api()`. If you do this, consider checking that your input data prototype has the same structure as your training data (perhaps with `hardhat::scream()`) and/or simulating data to avoid leaking PII via your deployed model.

Value

A new `vetiver_model` object.

Examples

```
cars_lm <- lm(mpg ~ ., data = mtcars)
vetiver_model(cars_lm, "cars_linear", pins::board_temp())
```

`vetiver_pin_metrics` *Update model metrics over time for monitoring*

Description

These three functions can be used for model monitoring (such as in a monitoring dashboard):

- `vetiver_compute_metrics()` computes metrics (such as accuracy for a classification model or RMSE for a regression model) at a chosen time aggregation `period`
- `vetiver_pin_metrics()` updates an existing pin storing model metrics over time
- `vetiver_plot_metrics()` creates a plot of metrics over time

Usage

```
vetiver_pin_metrics(
  board,
  df_metrics,
  metrics_pin_name,
  .index = .index,
  overwrite = FALSE,
  type = NULL,
  ...
)
```

Arguments

`board`	A pin board, created by `board_folder()`, `board_rsconnect()`, `board_url()` or another `board_` function.
`df_metrics`	A tidy dataframe of metrics over time, such as created by

`metrics_pin_name`
: Pin name for where the *metrics* are stored (as opposed to where the model object is stored with `vetiver_pin_write()`).

`.index`
: The variable in `df_metrics` containing the aggregated dates or date-times (from `time_var` in `data`). Defaults to `.index`.

`overwrite`
: If `FALSE` (the default), error when the new metrics contain overlapping dates with the existing pin.If `TRUE`, overwrite any metrics for dates that exist both in the existing pin and new metrics with the *new* values.

`type`
: File type used to save metrics to disk. With the default `NULL`, uses the `type` of the existing pin. Options are "rds" and "arrow".

`...`
: Additional arguments passed on to methods for a specific board.

Details

Sometimes when you monitor a model at a given time aggregation, you may end up with dates in your new metrics (like `new_metrics` in the example) that are the same as dates in your existing aggregated metrics (like `original_metrics` in the example). This can happen if you need to re-run a monitoring report because something failed. With `overwrite = FALSE` (the default), `vetiver_pin_metrics()` will error when there are overlapping dates. With `overwrite = TRUE`, `vetiver_pin_metrics()` will replace such metrics with the new values. You probably want `FALSE` for interactive use and `TRUE` for dashboards or reports that run on a schedule.

You can initially create your pin with `type = "arrow"` or the default (`type = "rds"`). `vetiver_pin_metrics()` will update the pin using the same `type` by default.

Value

A dataframe of metrics.

See Also

`vetiver_compute_metrics()`, `vetiver_plot_metrics()`

Examples

```
library(dplyr)
library(parsnip)
data(Chicago, package = "modeldata")
Chicago <- Chicago %>% select(ridership, date, all_of(stations))
training_data <- Chicago %>% filter(date < "2009-01-01")
testing_data <- Chicago %>% filter(date >= "2009-01-01", date < "2011-01-01")
monitoring <- Chicago %>% filter(date >= "2011-01-01", date < "2012-12-31")
lm_fit <- linear_reg() %>% fit(ridership ~ ., data = training_data)

library(pins)
b <- board_temp()

## before starting monitoring, initiate the metrics and pin
## (for example, with the testing data):
```

```
original_metrics <-
    augment(lm_fit, new_data = testing_data) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
pin_write(b, original_metrics, "lm_fit_metrics", type = "arrow")

## to continue monitoring with new data, compute metrics and update pin:
new_metrics <-
    augment(lm_fit, new_data = monitoring) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
vetiver_pin_metrics(b, new_metrics, "lm_fit_metrics")
```

`vetiver_pin_write` *Read and write a trained model to a board of models*

Description

Use `vetiver_pin_write()` to pin a trained model to a board of models, along with an input prototype for new data and other model metadata. Use `vetiver_pin_read()` to retrieve that pinned object.

Usage

```
vetiver_pin_write(board, vetiver_model, ...)

vetiver_pin_read(board, name, version = NULL)
```

Arguments

`board`	A pin board, created by `board_folder()`, `board_rsconnect()`, `board_url()` or another `board_` function.
`vetiver_model`	A deployable `vetiver_model()` object
`...`	Additional arguments passed on to methods for a specific board.
`name`	Pin name.
`version`	Retrieve a specific version of a pin. Use `pin_versions()` to find out which versions are available and when they were created.

Details

These functions read and write a `vetiver_model()` pin on the specified `board` containing the model object itself and other elements needed for prediction, such as the model's input data prototype or which packages are needed at prediction time. You may use `pins::pin_read()` or `pins::pin_meta()` to handle the pin, but `vetiver_pin_read()` returns a `vetiver_model()` object ready for deployment.

Value

`vetiver_pin_read()` returns a `vetiver_model()`; `vetiver_pin_write()` returns the name of the new pin, invisibly.

Examples

```
library(pins)
model_board <- board_temp()

cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(model_board, v)
model_board

vetiver_pin_read(model_board, "cars_linear")

# can use `version` argument to read a specific version:
pin_versions(model_board, "cars_linear")
```

---

vetiver_plot_metrics *Plot model metrics over time for monitoring*

Description

These three functions can be used for model monitoring (such as in a monitoring dashboard):

- `vetiver_compute_metrics()` computes metrics (such as accuracy for a classification model or RMSE for a regression model) at a chosen time aggregation `period`
- `vetiver_pin_metrics()` updates an existing pin storing model metrics over time
- `vetiver_plot_metrics()` creates a plot of metrics over time

Usage

```
vetiver_plot_metrics(
  df_metrics,
  .index = .index,
  .estimate = .estimate,
  .metric = .metric,
  .n = .n
)
```

Arguments

`df_metrics`	A tidy dataframe of metrics over time, such as created by
`.index`	The variable in `df_metrics` containing the aggregated dates or date-times (from `time_var` in `data`). Defaults to `.index`.

`.estimate`	The variable in `df_metrics` containing the metric estimate. Defaults to `.estimate`.
`.metric`	The variable in `df_metrics` containing the metric type. Defaults to `.metric`.
`.n`	The variable in `df_metrics` containing the number of observations used for estimating the metric.

Value

A `ggplot2` object.

See Also

`vetiver_compute_metrics()`, `vetiver_pin_metrics()`

Examples

```
library(dplyr)
library(parsnip)
data(Chicago, package = "modeldata")
Chicago <- Chicago %>% select(ridership, date, all_of(stations))
training_data <- Chicago %>% filter(date < "2009-01-01")
testing_data <- Chicago %>% filter(date >= "2009-01-01", date < "2011-01-01")
monitoring <- Chicago %>% filter(date >= "2011-01-01", date < "2012-12-31")
lm_fit <- linear_reg() %>% fit(ridership ~ ., data = training_data)

library(pins)
b <- board_temp()

## before starting monitoring, initiate the metrics and pin
## (for example, with the testing data):
original_metrics <-
    augment(lm_fit, new_data = testing_data) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
pin_write(b, original_metrics, "lm_fit_metrics", type = "arrow")

## to continue monitoring with new data, compute metrics and update pin:
new_metrics <-
    augment(lm_fit, new_data = monitoring) %>%
    vetiver_compute_metrics(date, "week", ridership, .pred, every = 4L)
vetiver_pin_metrics(b, new_metrics, "lm_fit_metrics")

library(ggplot2)
vetiver_plot_metrics(new_metrics) +
    scale_size(range = c(2, 4))
```

`vetiver_ptype.train` *Create a vetiver input data prototype*

Description

Optionally find and return an input data prototype for a model.

Usage

```
## S3 method for class 'train'
vetiver_ptype(model, ...)

## S3 method for class 'gam'
vetiver_ptype(model, ...)

## S3 method for class 'glm'
vetiver_ptype(model, ...)

## S3 method for class 'lm'
vetiver_ptype(model, ...)

## S3 method for class 'Learner'
vetiver_ptype(model, ...)

vetiver_ptype(model, ...)

## Default S3 method:
vetiver_ptype(model, ...)

vetiver_create_ptype(model, save_ptype, ...)

## S3 method for class 'ranger'
vetiver_ptype(model, ...)

## S3 method for class 'model_stack'
vetiver_ptype(model, ...)

## S3 method for class 'workflow'
vetiver_ptype(model, ...)

## S3 method for class 'xgb.Booster'
vetiver_ptype(model, ...)
```

Arguments

`model`	A trained model, such as an `lm()` model or a tidymodels `workflows::workflow()`.

...	Other method-specific arguments passed to `vetiver_ptype()` to compute an input data prototype, such as `ptype_data` (a sample of training features).
save_ptype	Should an input data prototype be stored with the model? The options are `TRUE` (the default, which stores a zero-row slice of the training data), `FALSE` (no input data prototype for visual documentation or checking), or a dataframe to be used for both checking at prediction time *and* examples in API visual documentation.

Details

These are developer-facing functions, useful for supporting new model types. A `vetiver_model()` object optionally stores an input data prototype for checking at prediction time.

- The default for `save_ptype`, `TRUE`, finds an input data prototype (a zero-row slice of the training data) via `vetiver_ptype()`.
- `save_ptype = FALSE` opts out of storing any input data prototype.
- You may pass your own data to `save_ptype`, but be sure to check that it has the same structure as your training data, perhaps with `hardhat::scream()`.

Value

A `vetiver_ptype` method returns a zero-row dataframe, and `vetiver_create_ptype()` returns either such a zero-row dataframe, `NULL`, or the dataframe passed to `save_ptype`.

Examples

```
cars_lm <- lm(mpg ~ cyl + disp, data = mtcars)

vetiver_create_ptype(cars_lm, TRUE)

## calls the right method for `model` via:
vetiver_ptype(cars_lm)

## can also turn off `ptype`
vetiver_create_ptype(cars_lm, FALSE)

## some models require that you pass in training features
cars_rf <- ranger::ranger(mpg ~ ., data = mtcars)
vetiver_ptype(cars_rf, ptype_data = mtcars[,-1])
```

---

vetiver_type_convert *Convert new data at prediction time using input data prototype*

Description

This is a developer-facing function, useful for supporting new model types. At prediction time, new observations typically must be checked and sometimes converted to the data types from training time.

Usage

```
vetiver_type_convert(new_data, ptype)
```

Arguments

new_data	New data for making predictions, such as a data frame.
ptype	An input data prototype, such as a 0-row slice of the training data

Value

A converted dataframe

Examples

```
library(tibble)
training_df <- tibble(x = as.Date("2021-01-01") + 0:9,
                      y = LETTERS[1:10], z = letters[11:20])
training_df

ptype <- vctrs::vec_slice(training_df, 0)
vetiver_type_convert(tibble(x = "2021-02-01", y = "J", z = "k"), ptype)

## unsuccessful conversion generates an error:
try(vetiver_type_convert(tibble(x = "potato", y = "J", z = "k"), ptype))

## error for missing column:
try(vetiver_type_convert(tibble(x = "potato", y = "J"), ptype))
```

---

vetiver_write_docker *Write a Dockerfile for a vetiver model*

Description

After creating a Plumber file with vetiver_write_plumber(), use vetiver_write_docker() to create a Dockerfile plus a vetiver_renv.lock file for a pinned vetiver_model().

Usage

```
vetiver_write_docker(
  vetiver_model,
  plumber_file = "plumber.R",
  path = ".",
  lockfile = "vetiver_renv.lock",
  rspm = TRUE,
  port = 8000,
  expose = TRUE
)
```

Arguments

vetiver_model	A deployable vetiver_model() object
plumber_file	A path for your Plumber file, created via vetiver_write_plumber(). Defaults to plumber.R in the working directory.
path	A path to write the Dockerfile and renv.lock lockfile, capturing the model's package dependencies. Defaults to the working directory.
lockfile	The generated lockfile in path. Defaults to "vetiver_renv.lock".
rspm	A logical to use the RStudio Public Package Manager for renv::restore() in the Docker image. Defaults to TRUE.
port	The server port for listening: a number such as 8080 or an expression like 'as.numeric(Sys.getenv("PORT"))' when the port is injected as an environment variable.
expose	Add EXPOSE to the Dockerfile? This is helpful for using Docker Desktop but does not work with an expression for port.

Value

The content of the Dockerfile, invisibly.

Examples

```
library(pins)
tmp_plumber <- tempfile()
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)
vetiver_write_plumber(b, "cars_linear", file = tmp_plumber)

## default port
vetiver_write_docker(v, tmp_plumber, tempdir())
## port from env variable
vetiver_write_docker(v, tmp_plumber, tempdir(),
                     port = 'as.numeric(Sys.getenv("PORT"))',
                     expose = FALSE)
```

---

vetiver_write_plumber *Write a deployable Plumber file for a vetiver model*

Description

Use vetiver_write_plumber() to create a plumber.R file for a vetiver_model() that has been versioned and stored via vetiver_pin_write().

Usage

```
vetiver_write_plumber(
  board,
  name,
  version = NULL,
  ...,
  file = "plumber.R",
  rsconnect = TRUE
)
```

Arguments

board	A pin board, created by board_folder(), board_rsconnect(), board_url() or another board_ function.
name	Pin name.
version	Retrieve a specific version of a pin. Use pin_versions() to find out which versions are available and when they were created.
...	Other arguments passed to vetiver_api() such as the endpoint path or prediction type.
file	A path to write the Plumber file. Defaults to plumber.R in the working directory. See plumber::plumb() for naming precedence rules.
rsconnect	Create a Plumber file with features needed for RStudio Connect? Defaults to TRUE.

Details

By default, this function will find and use the latest version of your vetiver model; the model API (when deployed) will be linked to that specific version. You can override this default behavior by choosing a specific version.

Value

The content of the plumber.R file, invisibly.

Examples

```
library(pins)
tmp <- tempfile()
b <- board_temp(versioned = TRUE)
cars_lm <- lm(mpg ~ ., data = mtcars)
v <- vetiver_model(cars_lm, "cars_linear")
vetiver_pin_write(b, v)

vetiver_write_plumber(b, "cars_linear", file = tmp)
```

Index