

stressor

This package is designed to allow the user to apply multiple machine learning methods by calling simple commands for data exploration. Python has a library, called PyCaret, which uses pipeline processes for fitting multiple models with a few lines of code. The `stressor` package uses the `reticulate` package to allow python to be run in R, giving access to the same tools that exist in python. One of the strengths of R is exploration. The `stressor` package gives you the freedom to explore the machine learning models side by side.

To get started, `stressor` requires that you have Python 3.8.10 installed on your computer. To install Python, please follow the instructions provided at:

<https://www.python.org/downloads/release/python-3810/>

Once Python is installed, you can install `stressor` from CRAN. For your convenience, we have attached `stressor` with the `library` statement to use the python features of `stressor`.

```
library(stressor)
```

Data Generation

It is convenient when testing new functions or algorithms to be able to generate toy data sets. With these toy data sets, we can choose the distribution of the parameters, of the error term, and the underlying model of the toy data set.

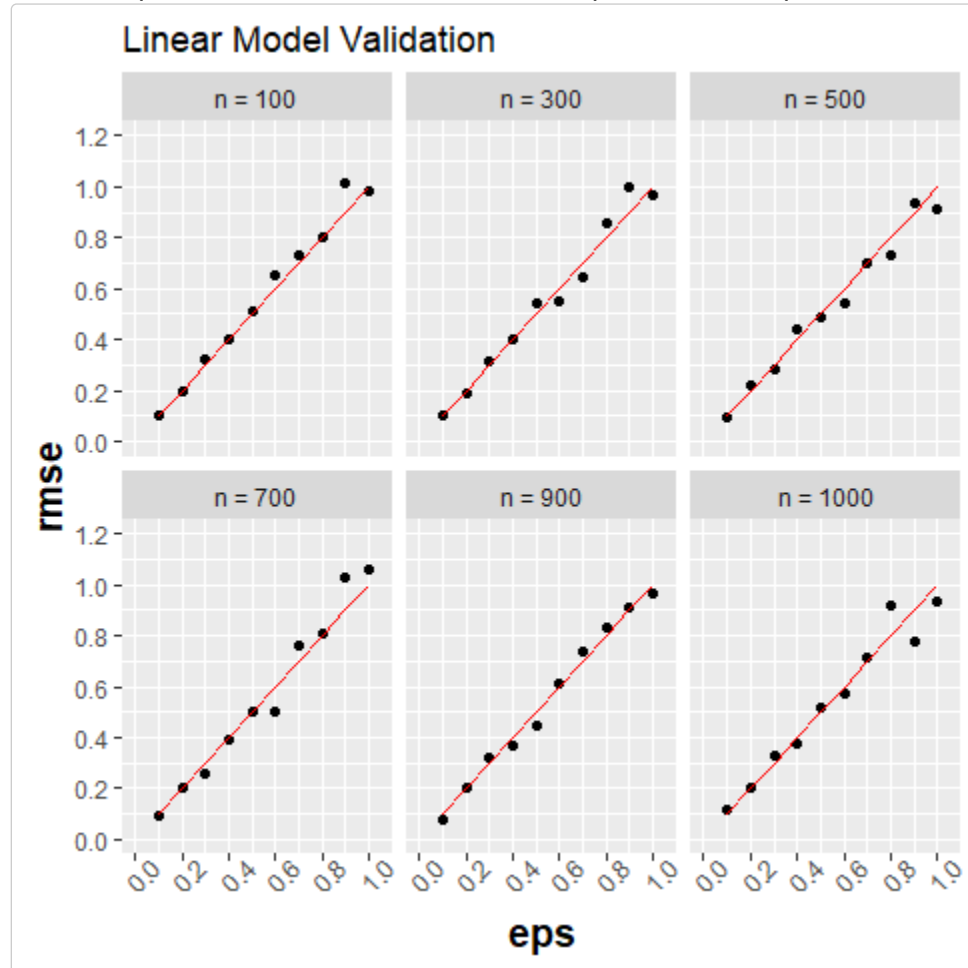
In this section, we will show an example of generating linear data with an epsilon and intercept that we chose. We will generate 500 observations from a linear model with five independent variables and a y-intercept of zero. Observations are simulated from this model assuming that the residuals follow a normal distribution with a mean of zero and a standard deviation of one. With respect to the variables chosen, each variable is sampled from a normal distribution with mean zero and standard deviation of one. For this case, we chose to let the coefficients on each term be one, as we wanted each independent variable to be equally weighted. When we create the response variable, Y, it is the sum of each independent variable plus an epsilon term that is sampled from a standard normal distribution.

```
set.seed(43421)
lm_data <- data_gen_lm(500, weight_vec = rep(1, 5), y_int = 0, resp_sd = 1)
head(lm_data)
```

#>	Y	V1	V2	V3	V4	V5
#> 1	1.5101730	0.9493875	-0.2231050	0.7501904	0.31629917	-0.41787475
#> 2	2.0124439	1.4844310	1.0737816	-1.8404303	0.85267167	-0.96389423
#> 3	2.6647624	-0.3505283	-0.3922640	0.7192181	0.05188511	1.60003509
#> 4	3.9270489	2.2945235	-0.8998011	0.1046142	1.45699275	1.01588132
#> 5	2.6975509	0.8574341	-0.9723329	-0.9897257	2.80821651	0.00363803
#> 6	0.8071714	0.7676524	-1.2666080	0.5582797	-0.80401673	0.12742990

Validation of Data Generation

Below is a visual of when we know the standard deviation of the epsilon term. We can show that our models fit the data if we are close to the theoretical error. In the graphic below, the black dots represent the value given the current epsilon that we are on. The red line represents the expected theoretical error.



Machine Learning Model Workflow

In this section, we will demonstrate a typical workflow using the functions of this package to explore the machine learning models (mlm) that are accessible through the `PyCaret` module in `python`. First, we need to create a virtual environment for the `PyCaret` module to exist in. The first time you run this code it will take some time (~ 5 min), as it needs to install the necessary modules into the virtual environment. Note that this virtual environment will be about 1 GB of space on the user's disk. `PyCaret` recommends that its library be used in a virtual environment. A virtual environment is a separate partition of `python` that can have a specific `python` version installed, as well as other `python` libraries. This enables the tools needed to be contained without disturbing the main version of `python` installed.

Once installed, the following message will be shown after you execute the code indicating that you are now using the virtual environment.

```
create_virtualenv()
```

See the [troubleshoot](#) section if other errors appear. The only time you will need to install a new environment is if you decide to delete a `stressor` environment and need to initiate a new one. You do not need to install a new environment for each `R` session, it is one and done. These environments are stored inside the `python` module on your computer.

To begin using, we need to create all the mlm. This may take a moment (< 3 min) the first time you run it, as the PyCaret module needs to be imported. Then depending on your data size it may take a moment (< 5 min for data <10,000) to fit the data. Note that console output will be shown and a progress bar will be displayed showing the progress of the fitting.

For reproducibility, we have set the seed again and have defined a new data set, and set the seed for the python side by passing the seed to the function. Here are the commands:

```
set.seed(43421)
lm_data <- data_gen_lm(1000)
# Split the data into a 80/20 split
indices <- split_data_prob(lm_data, .8)
train <- lm_data[indices, ]
test <- lm_data[!indices, ]
# Tune the models
mlm_lm <- mlm_regressor(Y ~ ., lm_data, sort_v = 'RMSE', seed = 43421)
```

Now, we can look at the initial training predictive accuracy measures such as RMSE. The `mlm_lm` is a list object where the first element is a list of all the models that were fitted. For example, if we were to pass these models back to PyCaret, they can be refitted or used again for predictions. The second element is a data frame for the initial values and the corresponding models. If you want to specify the models that are fitted, you can change the `fit_models` parameter – a character vector – specifying the models to be used. Also we can change how the models are sorted based upon the metrics listed which is given to the `sort_v` variable.

```
mlm_lm$pred_accuracy
```

```
#>
#>      Model      MAE      MSE      RMSE      R2      RMSLE
#> lr      Linear Regression 0.8345 1.0955 1.0429 0.8261 0.3664
#> ridge   Ridge Regression 0.8344 1.0955 1.0429 0.8261 0.3664
#> lar     Least Angle Regression 0.8345 1.0955 1.0429 0.8261 0.3664
#> br      Bayesian Ridge 0.8344 1.0955 1.0429 0.8261 0.3664
#> huber   Huber Regressor 0.8356 1.0976 1.0440 0.8259 0.3671
#> gbr     Gradient Boosting Regressor 1.0308 1.6365 1.2731 0.7425 0.4293
#> et      Extra Trees Regressor 0.9922 1.6474 1.2785 0.7406 0.4308
#> knn     K Neighbors Regressor 1.0231 1.6798 1.2936 0.7336 0.4390
#> lightgbm Light Gradient Boosting Machine 1.0432 1.7054 1.3013 0.7303 0.4331
#> rf      Random Forest Regressor 1.0448 1.7751 1.3253 0.7221 0.4406
#> ada     AdaBoost Regressor 1.1535 2.1419 1.4542 0.6656 0.4869
#> par     Passive Aggressive Regressor 1.2356 2.4503 1.5240 0.6015 0.4654
#> en      Elastic Net 1.4439 3.3031 1.8107 0.4877 0.6173
#> dt      Decision Tree Regressor 1.5140 3.6288 1.8966 0.4181 0.5561
#> omp     Orthogonal Matching Pursuit 1.8988 5.6044 2.3593 0.1243 0.6988
#> lasso   Lasso Regression 1.9174 5.7881 2.3973 0.1022 0.9362
#> llar    Lasso Least Angle Regression 1.9174 5.7881 2.3973 0.1022 0.9362
#> dummy   Dummy Regressor 2.0239 6.5123 2.5450 -0.0132 1.0254
#>
#>      MAPE TT (Sec)
#> lr      1.5432    0.009
#> ridge   1.5407    0.009
```

```
#> lar      1.5432    0.008
#> br       1.5405    0.009
#> huber    1.5460    0.010
#> gbr      1.8618    0.115
#> et       1.6469    0.133
#> knn      1.7811    0.009
#> lightgbm 2.0825    0.037
#> rf       1.6010    0.251
#> ada      1.5053    0.069
#> par      2.2490    0.009
#> en       1.0218    0.009
#> dt       2.2846    0.011
#> omp      2.1174    0.009
#> lasso    1.0678    0.009
#> llar     1.0678    0.008
#> dummy    1.0414    0.007
```

We pulled out a test validation set and we can currently check the accuracy measures of those predicted values, such as RMSE.

```
pred_lm <- predict(mlm_lm, test)
score(test$Y, pred_lm)
```

```
#>           rmse      mae      mse           r2      rmsle      mape
#> lr      0.9811816 0.7855389 0.9627174 0.8489721348 0.1575507 1.787733
#> ridge   0.9814217 0.7857800 0.9631885 0.8488982396 0.1576890 1.785325
#> lar     0.9811817 0.7855389 0.9627175 0.8489721181 0.1575508 1.787733
#> br      0.9814372 0.7857951 0.9632189 0.8488934604 0.1576978 1.785174
#> huber   0.9810649 0.7866087 0.9624883 0.8490080733 0.1576332 1.784634
#> gbr     1.1519822 0.8913621 1.3270630 0.7918148288 0.1819330 1.779195
#> et      1.2112827 0.9544208 1.4672058 0.7698296888 0.2063202 1.538019
#> knn     1.2961473 1.0304869 1.6799978 0.7364476099 0.2127031 1.447863
#> lightgbm 1.1740861 0.9132695 1.3784782 0.7837489759 0.1870156 1.385519
#> rf      1.2685104 0.9956673 1.6091187 0.7475668762 0.2078816 1.752251
#> ada     1.4734451 1.1557082 2.1710406 0.6594144709 0.2363207 1.464789
#> par     2.2243526 1.8216665 4.9477444 0.2238145292 0.2836395 3.536359
#> en      1.7919837 1.4126925 3.2112056 0.4962368798 0.2888262 1.161516
#> dt      1.9384331 1.4986607 3.7575229 0.4105324586 0.3019004 1.988530
#> omp     2.2605309 1.7922139 5.1100000 0.1983604118 0.3411484 1.932450
#> lasso   2.3843600 1.8606841 5.6851724 0.1081293000 0.3579984 1.037209
#> llar    2.3843599 1.8606841 5.6851724 0.1081293122 0.3579984 1.037209
#> dummy   2.5257071 1.9860032 6.3791965 -0.0007468551 0.3739615 1.065824
```

In comparison, we can fit this data using the `lm()` function and check the initial predictive accuracy with simple test data.

```
test_index <- split_data_prob(lm_data, .2)
test <- lm_data[test_index, ]
train <- lm_data[!test_index, ]
```

```
lm_test <- lm(Y ~ ., train)
lm_pred <- predict(lm_test, newdata = test)
lm_score <- score(test$Y, lm_pred)
lm_score
#>      RMSE      MAE      MSE      R2      RMSLE      MAPE
#> 0.9716537 0.7793268 0.9441110 0.8095243 0.1563934 1.0339178
```

As we look at this initial result, we see that there are some comparable models to the RMSE generated from `lm()` (which is 0.97 compared to 0.98 fitted by Huber Regressor). We see that the `mlm` outperforms the models that were fitted by `lm()`. However, it is not clear from this output alone whether the better performance observed from the `lm` model is statistically significant. A better practice would be performing a cross-validation.

In this code we are fitting the `mlm_lm` and `lm_test` to the `lm_data` using a 10 fold cross-validation.

First the ML models:

```
mlm_cv <- cv(mlm_lm, lm_data, n_folds = 10)
```

Then the `lm_test`:

```
lm_cv <- cv(lm_test, lm_data, n_folds = 10)
```

Now to compare the corresponding RMSE.

```
score(lm_data$Y, mlm_cv)
#>      RMSE      MAE      MSE      R2      RMSLE      MAPE
#> lr      2.364924 1.870219 5.592865 -0.9244870 0.2981139 2.776382
#> ridge   2.363766 1.869422 5.587389 -0.9226029 0.2979252 2.773871
#> lar      2.364924 1.870219 5.592865 -0.9244870 0.2981139 2.776382
#> br      2.363785 1.869435 5.587481 -0.9226344 0.2979288 2.773913
#> huber    2.353164 1.861261 5.537381 -0.9053951 0.2965040 2.761076
#> gbr      2.311607 1.828339 5.343527 -0.8386907 0.2905366 2.638235
#> et       2.269689 1.806269 5.151489 -0.7726111 0.2854914 2.508345
#> knn      2.335107 1.855689 5.452724 -0.8762651 0.2938159 2.636042
#> lightgbm 2.372964 1.869536 5.630957 -0.9375943 0.2985878 2.837477
#> rf       2.281818 1.817093 5.206693 -0.7916066 0.2870201 2.552018
#> ada      2.186354 1.754228 4.780145 -0.6448326 0.2771474 2.193652
#> par      2.576865 2.055230 6.640235 -1.2848838 0.2992653 3.583453
#> en       2.124765 1.711973 4.514628 -0.5534691 0.2731174 1.411179
#> dt       2.716927 2.166145 7.381690 -1.5400162 0.3503943 3.437621
#> omp      2.421582 1.945890 5.864059 -1.0178041 0.3030696 1.884166
#> lasso    2.355189 1.899553 5.546917 -0.9086763 0.3000909 1.058293
#> llar     2.355189 1.899553 5.546917 -0.9086763 0.3000909 1.058293
#> dummy    2.414994 1.947364 5.832195 -1.0068397 0.3066178 1.023737
score(lm_data$Y, lm_cv)
#>      RMSE      MAE      MSE      R2      RMSLE      MAPE
#> 1.0640627 0.8360287 1.1322295 0.8052017 0.1406658 1.5685498
```

We can see that the top five ML models are close in value to the linear model.

Real Data Example

We want to show how our functions apply to a real data example. We can simulate data, but it is never quite like observed data. The purpose of this data set is to show the use of the functions in this package – specifically cross-validation. This is crucial to show how these work in comparison to existing functions.

We will be using the Boston Housing Data from the `mlbench` package. There are two versions of this data, the second version includes a corrected `medv` value, standardizing the median income to USD 1000's. As some of the original data was missing. This data version also has had the town, tract, longitude and latitude added. For this analysis, we are ignoring spatial autocorrelation and therefore will be removing these variables from the analysis.

This next code chunk opens the cleaned Boston data set attached to this package and fits the initial machine learning models. It then displays the initial values from the first fit.

```
data(boston)
mlm_boston <- mlm_regressor(cmedv ~ ., boston)
mlm_boston$pred_accuracy
```

#>	Model	MAE	MSE	RMSE	R2	RMSLE
#> gbr	Gradient Boosting Regressor	2.1218	9.7524	3.0077	0.8615	0.1393
#> et	Extra Trees Regressor	2.1753	11.2063	3.1583	0.8453	0.1414
#> rf	Random Forest Regressor	2.2152	11.2131	3.2292	0.8441	0.1467
#> lightgbm	Light Gradient Boosting Machine	2.4390	13.9694	3.6343	0.8122	0.1570
#> ada	AdaBoost Regressor	2.7002	15.1353	3.7275	0.7950	0.1755
#> dt	Decision Tree Regressor	3.0190	20.6916	4.4073	0.7180	0.2007
#> lr	Linear Regression	3.3687	24.0099	4.7956	0.6858	0.2453
#> ridge	Ridge Regression	3.3493	24.0915	4.7963	0.6849	0.2513
#> lar	Least Angle Regression	3.4298	24.4397	4.8504	0.6785	0.2473
#> br	Bayesian Ridge	3.3931	24.7832	4.8727	0.6777	0.2589
#> huber	Huber Regressor	3.3622	27.7117	5.0719	0.6496	0.2931
#> en	Elastic Net	3.5681	27.9055	5.1803	0.6461	0.2562
#> lasso	Lasso Regression	3.6315	29.0143	5.2788	0.6328	0.2506
#> llar	Lasso Least Angle Regression	3.6315	29.0141	5.2788	0.6328	0.2506
#> knn	K Neighbors Regressor	3.9844	33.5862	5.7336	0.5557	0.2237
#> omp	Orthogonal Matching Pursuit	5.5777	62.2987	7.7159	0.2226	0.3140
#> dummy	Dummy Regressor	6.4549	78.6894	8.7760	-0.0148	0.3798
#> par	Passive Aggressive Regressor	7.1163	83.3044	8.9282	-0.1049	0.4482
#>	MAPE TT (Sec)					
#> gbr	0.1106	0.122				
#> et	0.1115	0.151				
#> rf	0.1148	0.263				
#> lightgbm	0.1197	0.056				
#> ada	0.1439	0.091				
#> dt	0.1550	0.035				
#> lr	0.1706	0.035				
#> ridge	0.1708	0.035				
#> lar	0.1737	0.035				
#> br	0.1730	0.035				

```
#> huber      0.1731      0.051
#> en         0.1724      0.035
#> lasso      0.1735      0.034
#> llar       0.1735      0.036
#> knn        0.1832      0.033
#> omp        0.2728      0.032
#> dummy      0.3508      0.032
#> par        0.3716      0.034
```

Observe the initial values for the Boston data set. Now compare these to the cross-validated values.

```
mlm_boston_cv <- cv(mlm_boston, boston, n_folds = 10)
mlm_boston_score <- score(boston$cmdev, mlm_boston_cv)
mlm_boston_score
```

```
#>          RMSE      MAE      MSE          R2      RMSLE      MAPE
#> gbr      2.880214 2.096367  8.295632  0.901413505 0.03826106 0.1097795
#> et       3.095790 2.038773  9.583918  0.886103331 0.04008861 0.1035976
#> rf       3.091200 2.145828  9.555518  0.886440846 0.04044387 0.1106439
#> lightgbm 3.223722 2.129938 10.392383  0.876495421 0.04176346 0.1082107
#> ada      3.582974 2.773172 12.837703  0.847434883 0.04837222 0.1497296
#> dt       4.485803 2.837352 20.122431  0.760862124 0.05795620 0.1437426
#> lr       4.908502 3.451202 24.093388  0.713670689 0.06492690 0.1745373
#> ridge    4.931873 3.439230 24.323376  0.710937485 0.06533195 0.1745591
#> lar      4.909012 3.469367 24.098402  0.713611106 0.06523700 0.1759787
#> br       5.006509 3.486602 25.065131  0.702122364 0.06636305 0.1770225
#> huber    5.584459 3.587230 31.186187  0.629378842 0.07572677 0.1812409
#> en       5.328285 3.710624 28.390621  0.662601752 0.06863061 0.1788665
#> lasso    5.386974 3.754152 29.019492  0.655128160 0.06925798 0.1803929
#> llar     5.386960 3.754145 29.019338  0.655129997 0.06925790 0.1803927
#> knn      5.836711 4.009170 34.067194  0.595140547 0.07285509 0.1819197
#> omp      8.116394 5.873471 65.875855  0.217121821 0.10323051 0.2868320
#> dummy    9.183814 6.643323 84.342437 -0.002337712 0.11942902 0.3630920
#> par      8.937533 6.676229 79.879500  0.050700482 0.12318012 0.3213036
```

Clustered cross-validation is subsetting the parameter space into groups that share similar attributes with one another. Therefore, if we train on those groups the other group should fit similarly across the test group.

Now, compare to the clustered cross-validation:

```
mlm_boston_clust_cv <- cv(mlm_boston, boston, n_folds = 10, k_mult = 5)
mlm_boston_clust_score <- score(boston$cmdev, mlm_boston_clust_cv)
mlm_boston_clust_score
```

```
#>          RMSE      MAE      MSE          R2      RMSLE      MAPE
#> gbr      3.752646 2.722356 14.08235  0.8326433 0.04915424 0.1408009
#> et       3.665735 2.566496 13.43761  0.8403055 0.04730942 0.1309730
#> rf       4.154256 2.798413 17.25785  0.7949053 0.05368193 0.1450138
```

```
#> lightgbm 4.023057 2.752783 16.18499 0.8076553 0.05210606 0.1408062
#> ada      4.433633 3.332406 19.65710 0.7663922 0.05852669 0.1778279
#> dt       5.394398 3.608300 29.09953 0.6541770 0.06953791 0.1882557
#> lr       5.879360 4.278917 34.56687 0.5892023 0.07986385 0.2287874
#> ridge    5.741469 4.099545 32.96447 0.6082455 0.07815901 0.2207254
#> lar      6.092678 4.370884 37.12072 0.5588520 0.08399236 0.2395658
#> br       5.808605 4.075319 33.73989 0.5990303 0.07884975 0.2192994
#> huber    6.444105 4.469043 41.52649 0.5064932 0.08726243 0.2302259
#> en       5.900504 4.247760 34.81595 0.5862423 0.07639839 0.2110481
#> lasso    6.105984 4.411099 37.28304 0.5569229 0.07907545 0.2195190
#> llar     6.106000 4.411162 37.28323 0.5569207 0.07907643 0.2195278
#> knn      8.027975 5.668933 64.44838 0.2340862 0.10082338 0.2561170
#> omp      8.542734 6.256828 72.97830 0.1327154 0.10913893 0.3073734
#> dummy    9.640141 7.050797 92.93233 -0.1044212 0.12579351 0.3862600
#> par      21.356794 14.663207 456.11264 -4.4205085 0.18068971 0.8859428
```

What we notice about this result is when we ignore spatial autocorrelation and we compare the 10 fold cross-validation with the clustered cross-validation, we see a general improvement in the values. This suggests that maybe there is some other underlying factors, i.e. spatial relationships.

The power to be able to explore is a compliment to the purpose of R. With `stressor`, you are able to fit multiple machine learning models with a few lines of code and perform 10 fold cross-validation and clustered cross-validation. With a simple command, you can return the values from the predictions.

Troubleshooting

When initiating the virtual environment, you may receive some errors or warnings. `reticulate` has done a nice job with the error handling of initiating the virtual environments. `reticulate` is a package in R that handles the connection between R and python.

For MacOS and Linux, please note that the `create_virtualenv()` function will not work unless you have `cmake`. `lightgbm` requires this compiler and they have detailed instructions of how to install it, see [here](#).

If your system is not recognizing the python path that you have, you will need to add it to your system variables, or specify initially the python path that `create_virtualenv()` needs to use. If you are still having trouble getting the virtual environment to start you can use `reticulate`'s function `reticulate::use_virtualenv()`. It also helps sometimes to unset the `RETICULATE_PYTHON` variable. Also note that if the environment has python objects in it the user will have to clear them to restart the `reticulate` python version.

If you receive a warning that says

“Warning Message: Previous request to `use_python()` ... will be ignored. It is superseded by request to `use_python()`”

If the second `use_python` command has the matching virtual environment you can ignore this warning and continue with your analysis.

If you receive an error stating

ERROR: The requested version of Python ... cannot be used, as another version of Python ... has already been initialized. Please restart the R session if you need to attach `reticulate` to a different version of Python.

If this error appears, restart your R session and make sure to clear all python objects. Then run the `create_virtualenv()` function again. There should be no problems attaching it after that, as long as your

environment does not contain any Python objects.