Developing and Deploying Transparent and Reproducible Algorithms for Public Health

R: tidymodels & plumber

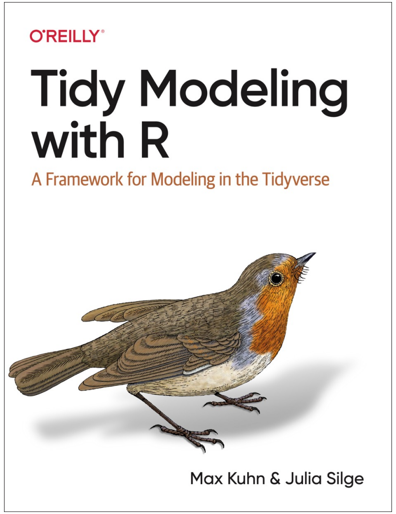
Doug Manuel
Juan Li
Wenshan Li
Kitty Chen

The Ottawa Hospital Research Institute | L'Hôpital d'Ottawa Institut de recherche

Affiliated with Affilié à uOttawa

TIDYMODELS

The tidymodels framework is a collection of packages for modeling and machine learning using tidyverse principles.
https://www.tidymodels.org/
https://www.tmwr.org/



PLUMBER

Plumber allows you to create a web API by merely decorating your existing R source code with roxygen2-like comments.
https://www.rplumber.io/

TIDYMODELS
Main packages

https://www.tidymodels.org/packages/

tidymodels — The meta-package

rsample — Data splitting

recipes — Data pre-processing

parsnip — Model interface

yardstick — Model performance

tune — Hyperparameter tuning of your model and pre-processing steps

dials — Creates and manages tuning parameters and parameter grids

workflows — Workflows

broom — Reporting

TIDYMODELS
Main packages

https://www.tidymodels.org/packages/

tidymodels — The meta-package

stacks — Model ensemble
To integrate predictions from many models

finetune — Extra functions for tuning

rsample — Data splitting
Train/test split, (nested) cross validation (CV) etc.

parsnip — Model interface

tune — Hyperparameter tuning
of your model and pre-processing steps

recipes — Data pre-processing

yardstick — Model performance

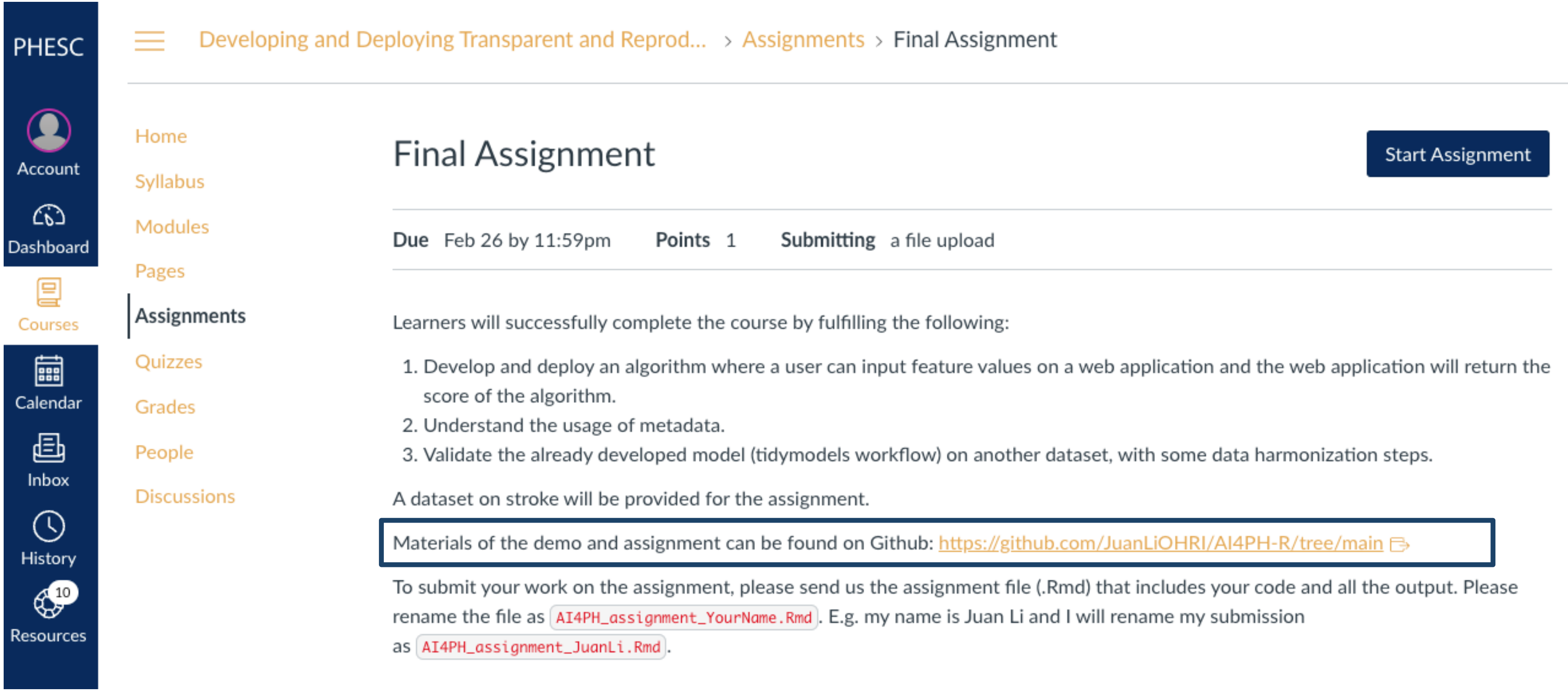
dials — Creates and manages tuning parameters and parameter grids

workflowsets
The goal of workflowsets is to allow users to create and easily fit a large number of models. workflowsets can create a *workflow set* that holds multiple workflow objects.

workflows — Workflows
bundle your pre-processing, modeling, and post-processing together

broom — Reporting
converts the information in common statistical R objects into user-friendly, predictable formats

Demo and assignment overview



Demo and assignment overview–
The Github repository (https://github.com/JuanLiOHRI/AI4PH-R/tree/main)

The demo code

The assignment code

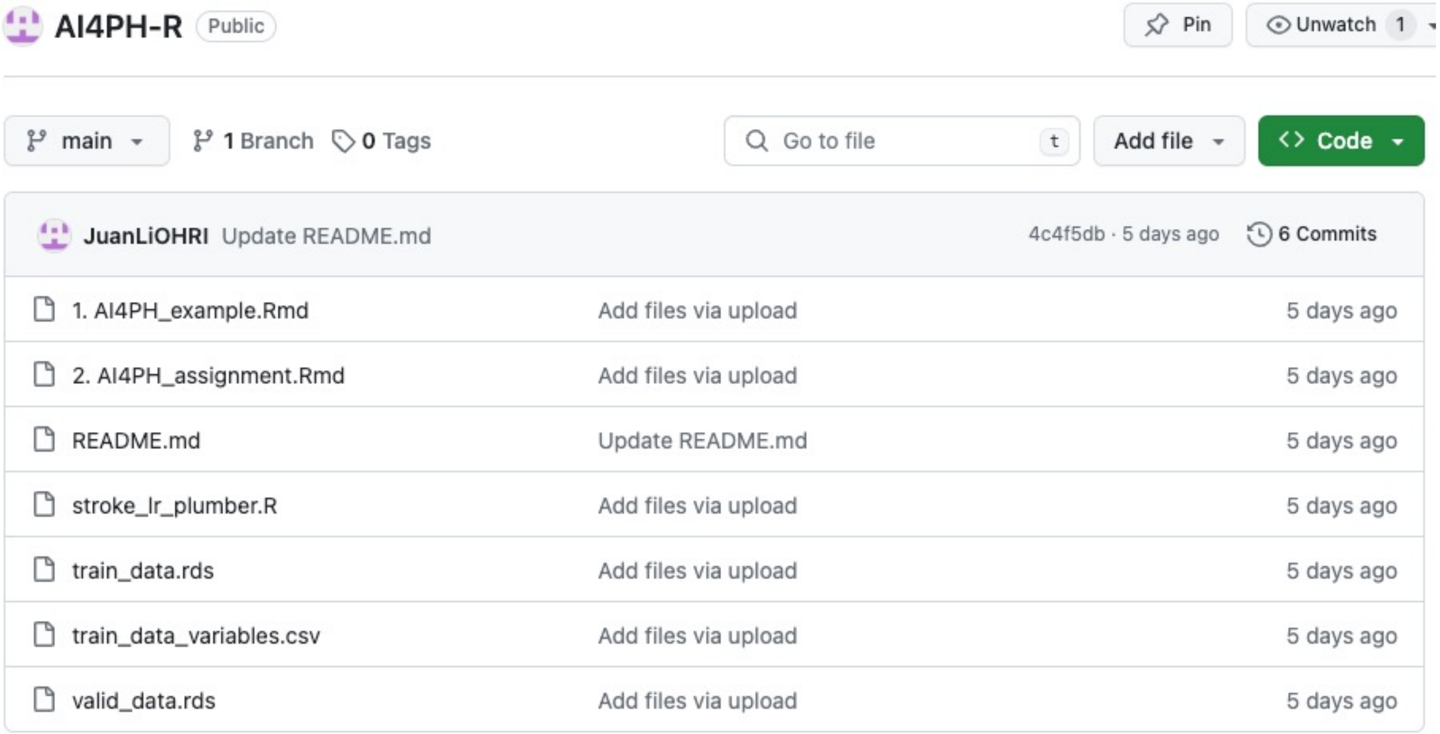
The Readme file

The plumber code

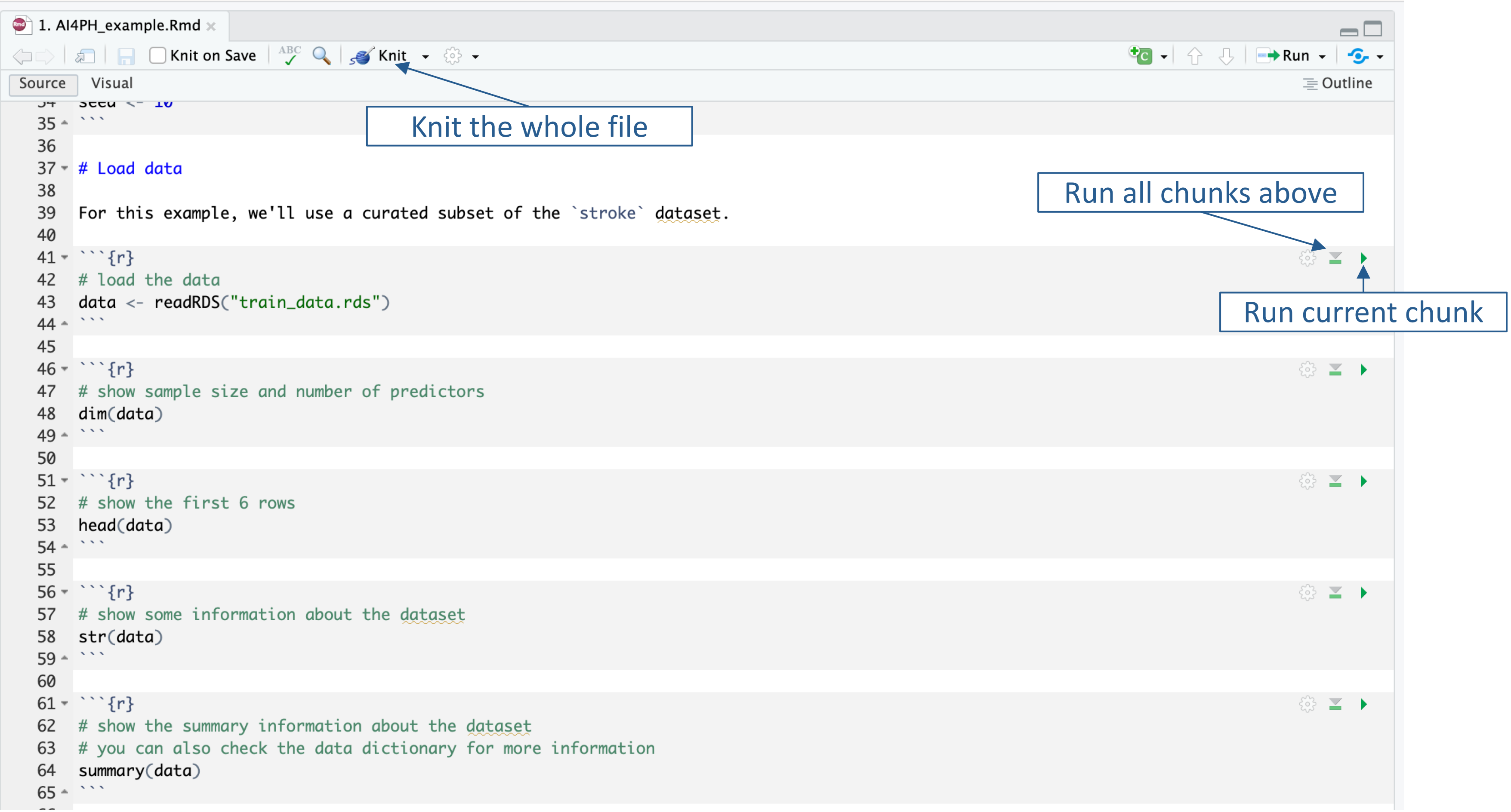
Data for the demo

The metadata

Data for the assignment



Preparation - R markdown file (.Rmd)



Preparation - Pipe operator: %>%

You can use the pipe operator (**%>%**) in R to “pipe” together a sequence of operations. This operator is most commonly used in R to perform a sequence of operations on a data frame.

The basic syntax for the pipe operator is:

```
df %>%
  operation1 %>%
  operation2 %>%
  operation3
  ...
```

this is equivalent to

```
df_step1 <- operation1(df,…)
df_step2 <- operation2(df_step1, …)
df_step3 <- operation3(df_step2, …)
...
```

The pipe operator simply feeds the results of one operation into the next operation below it. The advantage of using the pipe operator is that it makes code extremely easy to read.

https://www.statology.org/pipe-in-r/

TIDYMODELS (demo): a subset of the stroke data

```
> data <- readRDS("train_data.rds")
> dim(data)
[1] 4066   11
> head(data)
  gender      age hypertension heart_disease ever_married      work_type Residence_type avg_glucose_level      bmi   smoking_status stroke
1      M 49.33984          Yes            No          Yes    Private job          Rural         179.89718 27.84435     Never smoked      0
2      F 50.10903           No            No          Yes    Private job          Urban         136.81856 26.68095 Currently smokes      0
3      F 76.21449           No            No           No  Self-employed          Urban         201.03038 42.08255     Never smoked      0
5      F 66.76548          Yes            No          Yes    Private job          Urban          88.45523 25.77091     Never smoked      0
6      M 43.14164          Yes            No           No  Self-employed          Urban         249.41379 34.42329 Currently smokes      0
8      F 69.54896           No            No          Yes Government job          Urban          94.31868 28.52712  Formerly smoked      0
```

variable	role	type	min	max	label
gender	predictor	Categorical			F; M
age	predictor	Continuous	40	100	
hypertension	predictor	Categorical			No; Yes
heart_disease	predictor	Categorical			No; Yes
ever_married	predictor	Categorical			No; Yes
work_type	predictor	Categorical			Self-employed; Private job; Government job
Residence_type	predictor	Categorical			Rural; Urban
avg_glucose_level	predictor	Continuous	0	310	
bmi	predictor	Continuous	9	73	
smoking_status	predictor	Categorical			Never smoked; Formerly smoked; Currently smokes
stroke	outcome	Categorical			0; 1

TIDYMODELS (demo): data splitting

```
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()   # retrieve test data
```

TIDYMODELS (demo): model initializing

```r
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()   # retrieve test data
```

```r
# Initialize a logistic regression object
lr_spec <- logistic_reg() %>%
  # Set the model engine
  set_engine('glm') %>%
  # Set the model mode
  set_mode('classification')
```

TIDYMODELS (demo): data preprocessing

rsample

```
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()  # retrieve test data
```

parsnip

```
# Initialize a logistic regression object
lr_spec <- logistic_reg() %>%
  # Set the model engine
  set_engine('glm') %>%
  # Set the model mode
  set_mode('classification')
```

recipes

```
# Define the data preprocessing recipes
lr_recipe <-
  # define the formula
  recipe(stroke ~ ., data = stroke_train) %>%
  # create dummy variables for all the categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  # center and scale all numeric variables
  step_normalize(all_predictors())
```

TIDYMODELS (demo): define and train the workflow (model with pre-processing steps)

```r
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()   # retrieve test data
```

```r
# Define the data preprocessing recipes
lr_recipe <-
  # define the formula
  recipe(stroke ~ ., data = stroke_train) %>%
  # create dummy variables for all the categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  # center and scale all numeric variables
  step_normalize(all_predictors())
```

```r
# Initialize a logistic regression object
lr_spec <- logistic_reg() %>%
  # Set the model engine
  set_engine('glm') %>%
  # Set the model mode
  set_mode('classification')
```

```r
# Define the workflow
lr_workflow <-
  workflow() %>%
  add_model(lr_spec) %>%
  add_recipe(lr_recipe)
```

```r
# Train the model with preprocessing steps
lr_workflow_fit <- lr_workflow %>%
  fit(data = stroke_train)
```

Save the workflow

```{r}
saveRDS(lr_workflow_fit, "stroke_lr_workflow.rds")
```

TIDYMODELS (demo): obtaining the estimated coefficients

```
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()   # retrieve test data
```

```
# Initialize a logistic regression object
lr_spec <- logistic_reg() %>%
  # Set the model engine
  set_engine('glm') %>%
  # Set the model mode
  set_mode('classification')
```

```
# Define the data preprocessing recipes
lr_recipe <-
  # define the formula
  recipe(stroke ~ ., data = stroke_train) %>%
  # create dummy variables for all the categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  # center and scale all numeric variables
  step_normalize(all_predictors())
```

```
# Define the workflow
lr_workflow <-
  workflow() %>%
  add_model(lr_spec) %>%
  add_recipe(lr_recipe)
```

```
# Train the model with preprocessing steps
lr_workflow_fit <- lr_workflow %>%
  fit(data = stroke_train)
```

```
# Obtaining the estimated coefficients
tidy(lr_workflow_fit)
```

A tibble: 13 × 5

	term	estimate	std.error	statistic	p.value
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	(Intercept)	0.511	0.0401	12.7	4.45e-37
2	age	0.676	0.0449	15.1	3.25e-51
3	avg_glucose_level	0.320	0.0433	7.39	1.50e-13
4	bmi	-0.148	0.0420	-3.52	4.25e- 4
5	gender_M	0.0671	0.0403	1.67	9.58e- 2
6	hypertension_Yes	0.264	0.0418	6.31	2.70e-10
7	heart_disease_Yes	0.224	0.0464	4.83	1.34e- 6
8	ever_married_Yes	-0.0307	0.0408	-0.754	4.51e- 1
9	work_type_Private.job	-0.0620	0.0467	-1.33	1.85e- 1
10	work_type_Government.job	-0.0403	0.0462	-0.872	3.83e- 1
11	Residence_type_Urban	0.0324	0.0399	0.812	4.17e- 1
12	smoking_status_Formerly.smoked	0.0226	0.0431	0.524	6.00e- 1
13	smoking_status_Currently.smokes	0.00873	0.0420	0.208	8.35e- 1

```
# Obtaining the odds ratios
tidy(lr_workflow_fit, exponentiate = TRUE)
```

A tibble: 13 × 5

	term	estimate	std.error	statistic	p.value
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	(Intercept)	1.67	0.0401	12.7	4.45e-37
2	age	1.97	0.0449	15.1	3.25e-51
3	avg_glucose_level	1.38	0.0433	7.39	1.50e-13
4	bmi	0.862	0.0420	-3.52	4.25e- 4
5	gender_M	1.07	0.0403	1.67	9.58e- 2
6	hypertension_Yes	1.30	0.0418	6.31	2.70e-10
7	heart_disease_Yes	1.25	0.0464	4.83	1.34e- 6
8	ever_married_Yes	0.970	0.0408	-0.754	4.51e- 1
9	work_type_Private.job	0.940	0.0467	-1.33	1.85e- 1
10	work_type_Government.job	0.961	0.0462	-0.872	3.83e- 1
11	Residence_type_Urban	1.03	0.0399	0.812	4.17e- 1
12	smoking_status_Formerly.smoked	1.02	0.0431	0.524	6.00e- 1
13	smoking_status_Currently.smokes	1.01	0.0420	0.208	8.35e- 1

TIDYMODELS (demo): predicting on the test data

rsample

```r
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()   # retrieve test data
```

recipes

```r
# Define the data preprocessing recipes
lr_recipe <-
  # define the formula
  recipe(stroke ~ ., data = stroke_train) %>%
  # create dummy variables for all the categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  # center and scale all numeric variables
  step_normalize(all_predictors())
```

broom

```r
# Obtaining the estimated coefficients
tidy(lr_workflow_fit)
```

```r
# Obtaining the odds ratios
tidy(lr_workflow_fit, exponentiate = TRUE)
```

parsnip

```r
# Initialize a logistic regression object
lr_spec <- logistic_reg() %>%
  # Set the model engine
  set_engine('glm') %>%
  # Set the model mode
  set_mode('classification')
```

WORKFLOWS

```r
# Define the workflow
lr_workflow <-
  workflow() %>%
  add_model(lr_spec) %>%
  add_recipe(lr_recipe)
```

```r
# Train the model with preprocessing steps
lr_workflow_fit <- lr_workflow %>%
  fit(data = stroke_train)
```

```r
# The predicted classes
prediction_class <- lr_workflow_fit %>%
  predict(new_data = stroke_test,
          type = 'class')
```

```r
# The predicted probability of each class
prediction_prob <- lr_workflow_fit %>%
  predict(new_data = stroke_test,
          type = 'prob')
```

```r
# Combine test data with predictions
test_results <- stroke_test %>%
  bind_cols(prediction_class, prediction_prob)
```

TIDYMODELS (demo): model evaluation on the test data

```r
set.seed(seed) # set seed for reproducibility

# split the data into train and test datasets
stroke_split <- initial_split(data, prop = 0.8, strata = stroke)
stroke_train <- stroke_split %>% training()  # retrieve train data
stroke_test  <- stroke_split %>% testing()   # retrieve test data
```

```r
# Define the data preprocessing recipes
lr_recipe <-
  # define the formula
  recipe(stroke ~ ., data = stroke_train) %>%
  # create dummy variables for all the categorical predictors
  step_dummy(all_nominal_predictors()) %>%
  # center and scale all numeric variables
  step_normalize(all_predictors())
```

```r
# Obtaining the estimated coefficients
tidy(lr_workflow_fit)
```

```r
# Obtaining the odds ratios
tidy(lr_workflow_fit, exponentiate = TRUE)
```

```r
# Initialize a logistic regression object
lr_spec <- logistic_reg() %>%
  # Set the model engine
  set_engine('glm') %>%
  # Set the model mode
  set_mode('classification')
```

```r
# Define the workflow
lr_workflow <-
  workflow() %>%
  add_model(lr_spec) %>%
  add_recipe(lr_recipe)
```

```r
# Train the model with preprocessing steps
lr_workflow_fit <- lr_workflow %>%
  fit(data = stroke_train)
```

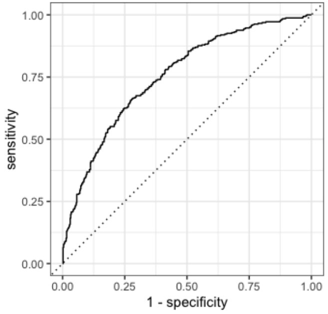
````
```{r, message=FALSE}
custom_metrics <- metric_set(sens, spec, roc_auc)

custom_metrics(test_results, truth = stroke, estimate = .pred_class, .pred_0)
```
````

A tibble: 3 × 3

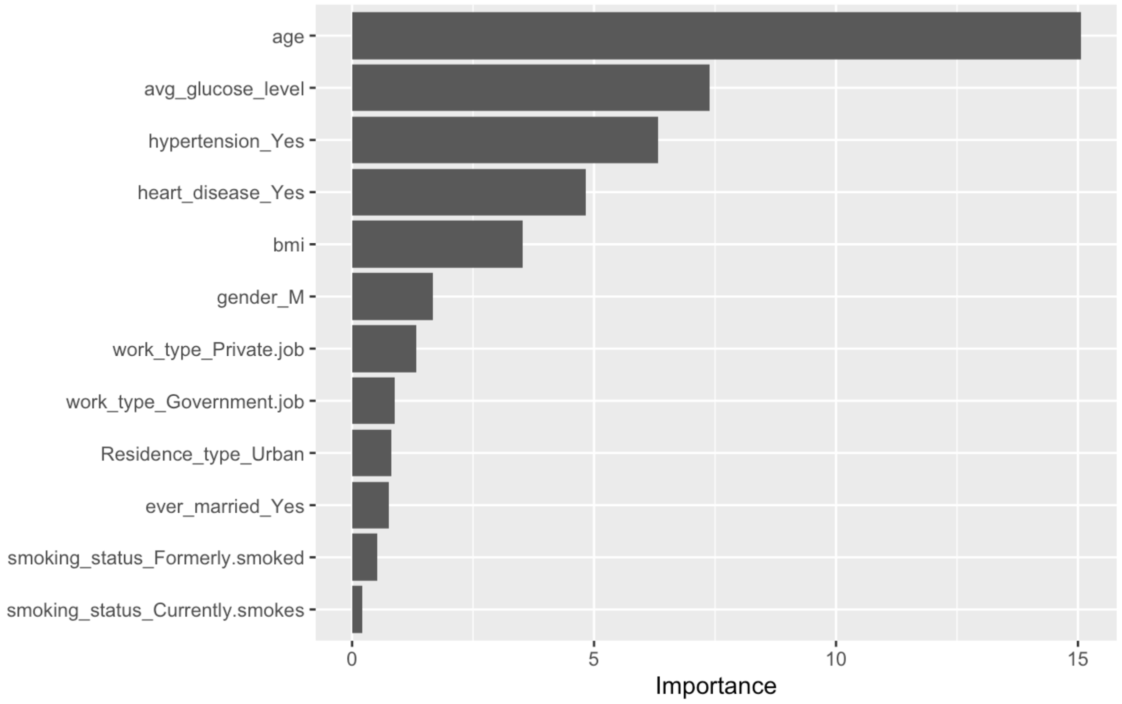
.metric <chr>	.estimator <chr>	.estimate <dbl>
sens	binary	0.5417957
spec	binary	0.8126273
roc_auc	binary	0.7533750

````
```{r, message=FALSE}
test_results %>% roc_curve(stroke, .pred_0) %>% autoplot()
```
````



TIDYMODELS (demo): feature importance

```
```{r, message=FALSE}
lr_workflow_fit %>%
  extract_fit_parsnip() %>%
  vip::vip(num_features = 20)
```
```



PLUMBER (demo)

stroke_lr_workflow.rds

The saved workflow (model with data preprocessing steps) that has been trained on the train data (and hopefully evaluated on the test data).

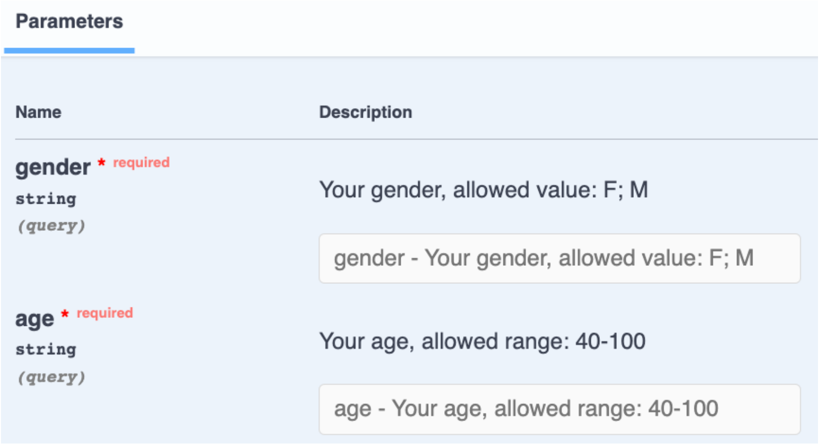
stroke_lr_plumber.R

The plumber script.

PLUMBER (demo)

```r
#* @param gender Your gender, allowed value: F; M
#* @param age Your age, allowed range: 40-100
#* @param hypertension Do you have hypertension? Allowed value: No; Yes
#* @param heart_disease Do you have heart disease? Allowed range: No; Yes
#* @param ever_married Have you ever married? Allowed range: No;Yes
#* @param work_type What kind of work you are doing or have done? Allowed value: Self-employed; Private job; Government job
#* @param Residence_type The type of your residence, allowed range: Rural; Urban
#* @param avg_glucose_level Average glucose level, allowed range: 0-310
#* @param bmi Body mass index, allowed range: 9-73
#* @param smoking_status Smoking, allowed value: Never smoked; Formerly smoked; Currently smokes
#* @get /predict/values

function(gender, age, hypertension, heart_disease, ever_married, work_type,
         Residence_type, avg_glucose_level, bmi, smoking_status) {
```



PLUMBER (demo)

```r
# read in the saved workflow object
workflow <- readRDS("stroke_lr_workflow.rds")

# assemble the inputs into a data frame
newdata <- data.frame(gender = factor(gender),
                      age = as.numeric(age),
                      hypertension = factor(hypertension),
                      heart_disease = factor(heart_disease),
                      ever_married = factor(ever_married),
                      work_type = factor(work_type),
                      Residence_type = factor(Residence_type),
                      avg_glucose_level = as.numeric(avg_glucose_level),
                      bmi = as.numeric(bmi),
                      smoking_status = factor(smoking_status)
)
```

stroke_lr_workflow.rds

PLUMBER (demo)

```r
# predict on the new data - class
prediction_class <- workflow %>%
  predict(new_data = newdata,
          type = 'class')

# predict on the new data - probability
prediction_prob <- workflow %>%
  predict(new_data = newdata,
          type = 'prob')

# report result
print(paste("The predicted class is: ", prediction_class$.pred_class, ", where 0 = no stroke; 1 = stroke",
            ". The predicted probability for class 0 is: ", round(prediction_prob[1],3),
            ". The predicted probability for class 1 is: ", round(prediction_prob[2],3), sep = ""))
```

Server response

Code	Details
200	Response body

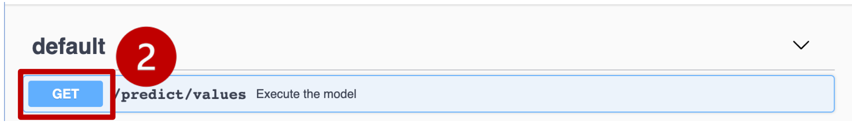

```
[
  "The predicted class is: 1, where 0 = no stroke; 1 = stroke. The predicted probability for class 0 is: 0.415. The predicted probability for class 1 is: 0.585"
]
```

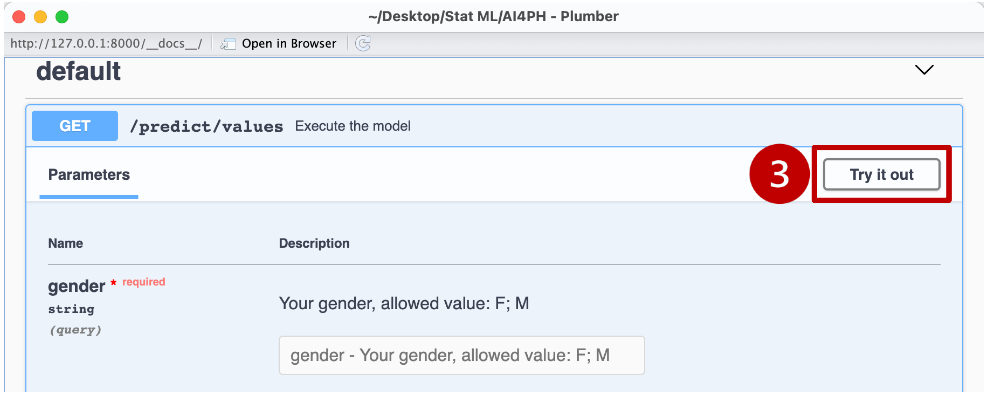

Download

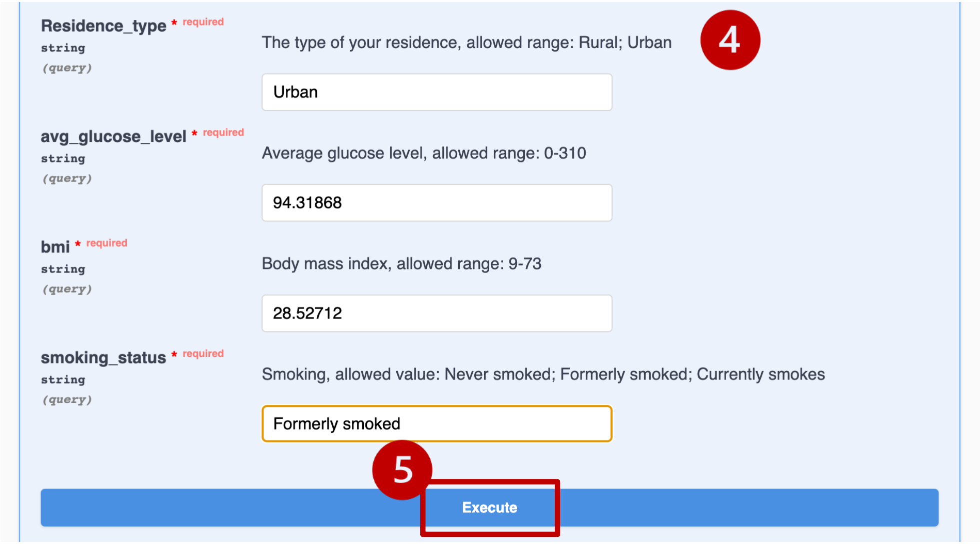
PLUMBER (demo)

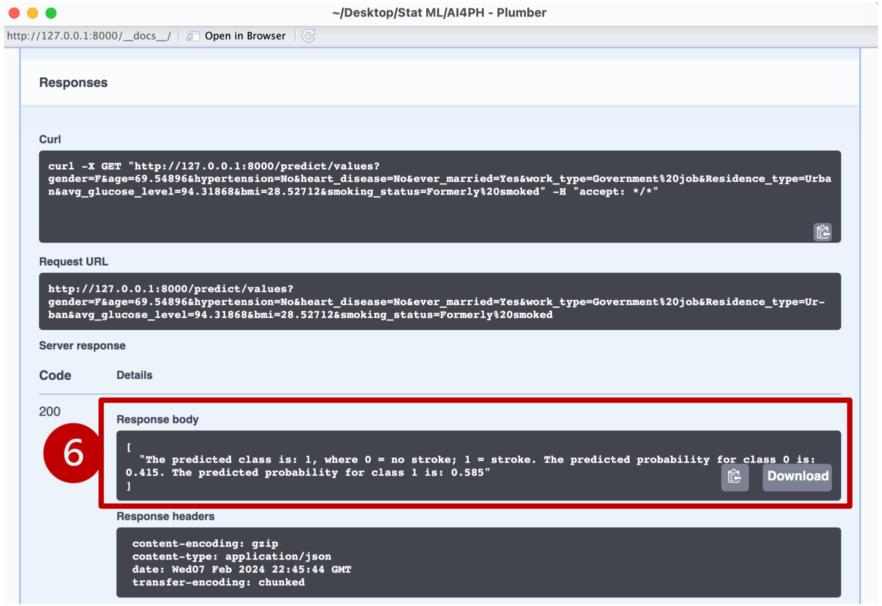
1

```{r}
# 'stroke_lr_plumber.R' is the location of the file shown above
pr("stroke_lr_plumber.R") %>%
  pr_run(port=8000)
```









Demo

Please copy and paste the below examples and see if the returned results are as expected:

Example 1

gender	F
age	69.54896
hypertension	No
heart_disease	No
ever_married	Yes
work_type	Government job
Residence_type	Urban
avg_glucose_level	94.31868
bmi	28.52712
smoking_status	Formerly smoked

===============

The observed class: 1

===============

The expected predicted class: 1

The expected probability for class 0: 0.415

The expected probability for class 1: 0.585

Example 2

gender	F
age	48.50831
hypertension	No
heart_disease	No
ever_married	Yes
work_type	Private job
Residence_type	Rural
avg_glucose_level	61.57483
bmi	27.60176
smoking_status	Never smoked

===============

The observed class: 0

===============

The expected predicted class: 0

The expected probability for class 0: 0.742

The expected probability for class 1: 0.258

Example 3

gender	M
age	68.96269
hypertension	No
heart_disease	No
ever_married	Yes
work_type	Self-employed
Residence_type	Urban
avg_glucose_level	77.88883
bmi	25.83863
smoking_status	Formerly smoked

===============

The observed class: 1

===============

The expected predicted class: 1

The expected probability for class 0: 0.371

The expected probability for class 1: 0.629

The final assessment

In this assignment, you will validate the model we developed in class using a different dataset: `valid_data.rds`. You will run into issues using this dataset as it is because this is a raw dataset without data harmonization, which means that some variables in this dataset are different from the harmonized dataset we used to train and evaluate the model. Your job here is to harmonize the validation data so that it's in the same format as the example data we used in class. You can refer to `train_data_variables.csv` to see the format in the harmonized train data.

All materials including this slide deck and the demo code can be found on Github: https://github.com/JuanLiOHRI/AI4PH-R/tree/main

The final assessment – How to submit

PHESC

Account

Dashboard

Courses

Calendar

Inbox

History

10

Resources

Developing and Deploying Transparent and Reprod... › Assignments › Final Assignment

Home

Syllabus

Modules

Pages

Assignments

Quizzes

Grades

People

Discussions

Final Assignment

Start Assignment

Due Feb 26 by 11:59pm **Points** 1 **Submitting** a file upload

Learners will successfully complete the course by fulfilling the following:

1. Develop and deploy an algorithm where a user can input feature values on a web application and the web application will return the score of the algorithm.
2. Understand the usage of metadata.
3. Validate the already developed model (tidymodels workflow) on another dataset, with some data harmonization steps.

A dataset on stroke will be provided for the assignment.

Materials of the demo and assignment can be found on Github: https://github.com/JuanLiOHRI/AI4PH-R/tree/main

To submit your work on the assignment, please send us the assignment file (.Rmd) that includes your code and all th rename the file as `AI4PH_assignment_YourName.Rmd`. E.g. my name is Juan Li and I will rename my submission as `AI4PH_assignment_JuanLi.Rmd`.

File Upload | Google Drive

Upload a file, or choose a file you've already uploaded.

Upload File | Use Webcam

+ Add Another File

Comments...

Cancel | Submit Assignment

THANK YOU

TIDYMODELS – other packages that might be useful

Some R objects become inconveniently large when saved to disk. The butcher package can reduce the size of those objects by removing the sub-components.

e.g. logistic regression model object that contains the training data. Especially in public health, where the sample size can be huge.

The tidyposterior package enables users to make formal statistical comparisons between models using resampling and Bayesian methods.

infer is a high-level API for tidyverse-friendly statistical inference.

The corrr package has tidy interfaces for working with correlation matrices.

https://www.tidymodels.org/packages/