

Data analysis & modeling report

The following required system packages are not installed:

- libglpk-dev [required by igraph]
- libx11-dev [required by clipr]
- pandoc [required by DataExplorer, knitr, rmarkdown]

The R packages depending on these system packages may fail to install.

An administrator can install these packages with:

- `sudo apt install libglpk-dev libx11-dev pandoc`
- The library is already synchronized with the lockfile.

Required packages

```
library(readr)
library(dplyr)
```

Adjuntando el paquete: 'dplyr'

The following objects are masked from 'package:stats':

`filter`, `lag`

The following objects are masked from 'package:base':

`intersect`, `setdiff`, `setequal`, `union`

```
library(tidyr)
library(DataExplorer)
library(tidymodels)
```

```
-- Attaching packages ----- tidymodels 1.3.0 --
```

v broom	1.0.8	v recipes	1.2.1
v dials	1.4.0	v rsample	1.3.0
v ggplot2	3.5.2	v tibble	3.2.1
v infer	1.0.7	v tune	1.3.0
v modeldata	1.4.0	v workflows	1.2.0
v parsnip	1.3.1	v workflowsets	1.1.0
v purrr	1.0.4	v yardstick	1.3.2

```
-- Conflicts ----- tidymodels_conflicts() --
```

```
x purrr::discard() masks scales::discard()
x dplyr::filter() masks stats::filter()
x dplyr::lag() masks stats::lag()
x yardstick::spec() masks readr::spec()
x recipes::step() masks stats::step()
```

```
library(glmnet)
```

Cargando paquete requerido: Matrix

Adjuntando el paquete: 'Matrix'

The following objects are masked from 'package:tidyr':

expand, pack, unpack

Loaded glmnet 4.1-8

```
library(vip)
```

Adjuntando el paquete: 'vip'

The following object is masked from 'package:utils':

vi

```
library(ranger)
library(readxl)
library(kernelshap)
library(shapviz)
```

Load dataset

```
database <- read_excel("database_2021_2024_con out.xlsx",
  col_types = c("date", "numeric", "text",
    "date", "numeric", "numeric", "numeric",
    "skip", "skip", "numeric", "numeric",
    "numeric", "numeric", "numeric",
    "numeric", "numeric"))

colnames(database)
```

```
[1] "Dia"                "Año"
[3] "Estación"          "Hora"
[5] "Temperatura (°C)"  "Humedad (%)"
[7] "Presión (hPa)"     "Velocidad de viento (m/s)"
[9] "CO (mg/m3)"        "NO (ug/m3)"
[11] "NO2 (ug/m3)"       "NOX (ug/m3)"
[13] "O3 (ug/m3)"        "PM10 (ug/m3)"
```

```
nrow(database)
```

```
[1] 1747
```

```
summary(database)
```

Dia		Año		Estación
Min.	:2021-07-27 00:00:00.00	Min.	:2021	Length:1747
1st Qu.	:2022-02-16 00:00:00.00	1st Qu.	:2022	Class :character
Median	:2023-01-12 00:00:00.00	Median	:2023	Mode :character

Mean	:2022-12-05 06:32:21.16	Mean	:2022
3rd Qu.	:2023-10-12 00:00:00.00	3rd Qu.	:2023
Max.	:2024-04-04 00:00:00.00	Max.	:2024

Hora	Temperatura (°C)	Humedad (%)
Min. :1899-12-31 00:00:00.00	Min. :-1.70	Min. : 3.00
1st Qu.:1899-12-31 06:00:00.00	1st Qu.:12.18	1st Qu.:33.00
Median :1899-12-31 13:00:00.00	Median :19.20	Median :46.00
Mean :1899-12-31 16:00:37.09	Mean :18.03	Mean :48.05
3rd Qu.:1899-12-31 19:00:00.00	3rd Qu.:24.00	3rd Qu.:62.00
Max. :1900-01-05 08:00:00.00	Max. :36.60	Max. :95.00

Presión (hPa)	Velocidad de viento (m/s)	CO (mg/m3)	NO (ug/m3)
Min. :902.0	Min. : 0.000	Min. :0.000	Min. : 0.020
1st Qu.:923.4	1st Qu.: 0.290	1st Qu.:0.700	1st Qu.: 4.955
Median :927.2	Median : 0.950	Median :1.130	Median : 17.390
Mean :927.1	Mean : 1.438	Mean :1.188	Mean : 32.230
3rd Qu.:930.5	3rd Qu.: 1.940	3rd Qu.:1.595	3rd Qu.: 47.910
Max. :942.5	Max. :18.060	Max. :3.460	Max. :417.340
NA's :160			

NO2 (ug/m3)	NOX (ug/m3)	O3 (ug/m3)	PM10 (ug/m3)
Min. : 1.28	Min. : 4.15	Min. : 0.00	Min. : 0.00
1st Qu.: 28.23	1st Qu.: 40.81	1st Qu.: 6.61	1st Qu.: 16.00
Median : 43.22	Median : 71.98	Median : 30.75	Median : 28.50
Mean : 43.80	Mean : 90.56	Mean : 58.83	Mean : 36.37
3rd Qu.: 54.23	3rd Qu.:127.17	3rd Qu.: 64.16	3rd Qu.: 48.50
Max. :131.05	Max. :559.91	Max. :509.65	Max. :318.00
			NA's :16

Cleaning

Delete columns

- Keep variables Temperatura, Humedad relativa, Presión atmosférica, Velocidad de viento, CO, NO, NO2, O3.
- Delete column NOX
- Delete NA

```
data <- database %>% select(-one_of(c("Estación","Dia","Año","Hora","NOX (ug/m3)")))
```

Clean the features' names

```
library(stringr)
```

Adjuntando el paquete: 'stringr'

The following object is masked from 'package:recipes':

fixed

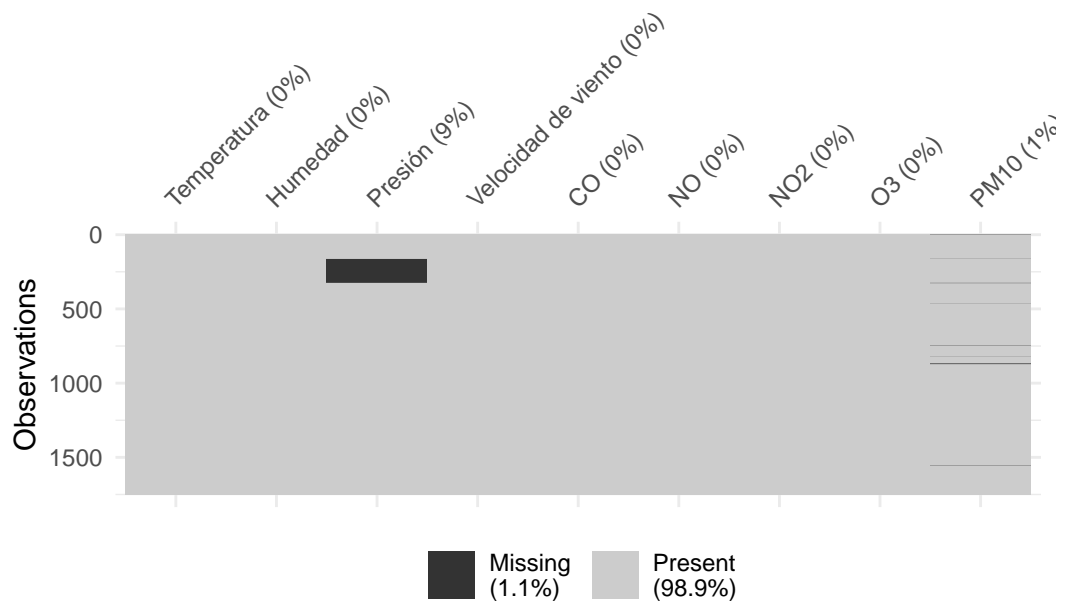
```
colnames(data) <- str_replace(colnames(data), pattern="\\s+\\((\\S+", "")
```

Missing values

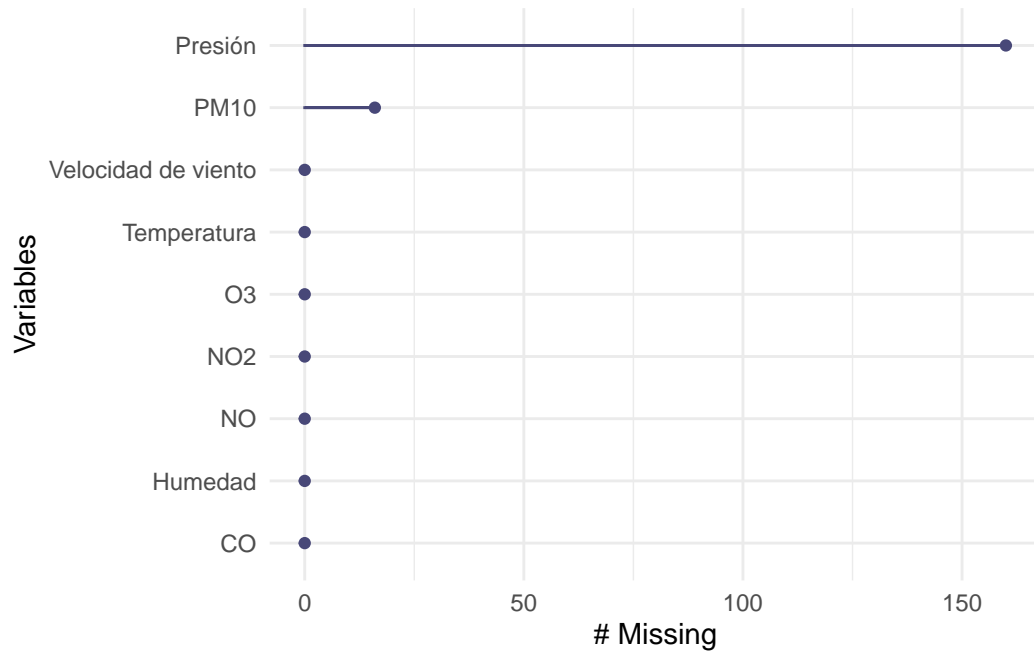
```
# visualizar los valores perdidos del dataset data. Instalar la libreria naniar si no está i
if(!require(naniar)){
  install.packages("naniar")
  library(naniar)
}
```

Cargando paquete requerido: naniar

```
vis_miss(data)
```



```
gg_miss_var(data)
```



```
data_imputed <- data %>%
  mutate(
    PM10 = ifelse(is.na(PM10), median(PM10, na.rm = TRUE), PM10),
    Presión = ifelse(is.na(Presión), median(Presión, na.rm = TRUE), Presión)
  )
```

- The dataset has 1747 samples (rows) and 9 variables (columns).
- The features (variables) are: Temperatura, Humedad, Presión, Velocidad de viento, CO, NO, NO2, O3, PM10.

Variable name

```
data <- data_imputed
colnames(data)
```

```
[1] "Temperatura"      "Humedad"          "Presión"
[4] "Velocidad de viento" "CO"               "NO"
[7] "NO2"              "O3"               "PM10"
```

- Translate names to English

```
colnames(data)[1:4] <- c("Temperature", "Humidity", "Pressure", "Wind velocity")
```

```
colnames(data)
```

```
[1] "Temperature"  "Humidity"      "Pressure"      "Wind velocity"
[5] "CO"           "NO"            "NO2"          "O3"
[9] "PM10"
```

- Data summary

```
summary(data)
```

Temperature	Humidity	Pressure	Wind velocity
Min. : -1.70	Min. : 3.00	Min. : 902.0	Min. : 0.000
1st Qu.: 12.18	1st Qu.: 33.00	1st Qu.: 924.0	1st Qu.: 0.290
Median : 19.20	Median : 46.00	Median : 927.2	Median : 0.950
Mean : 18.03	Mean : 48.05	Mean : 927.1	Mean : 1.438

3rd Qu.:24.00	3rd Qu.:62.00	3rd Qu.:930.1	3rd Qu.: 1.940
Max. :36.60	Max. :95.00	Max. :942.5	Max. :18.060
CO	NO	NO2	O3
Min. :0.000	Min. : 0.020	Min. : 1.28	Min. : 0.00
1st Qu.:0.700	1st Qu.: 4.955	1st Qu.: 28.23	1st Qu.: 6.61
Median :1.130	Median : 17.390	Median : 43.22	Median : 30.75
Mean :1.188	Mean : 32.230	Mean : 43.80	Mean : 58.83
3rd Qu.:1.595	3rd Qu.: 47.910	3rd Qu.: 54.23	3rd Qu.: 64.16
Max. :3.460	Max. :417.340	Max. :131.05	Max. :509.65
PM10			
Min. : 0.0			
1st Qu.: 16.0			
Median : 28.5			
Mean : 36.3			
3rd Qu.: 48.0			
Max. :318.0			

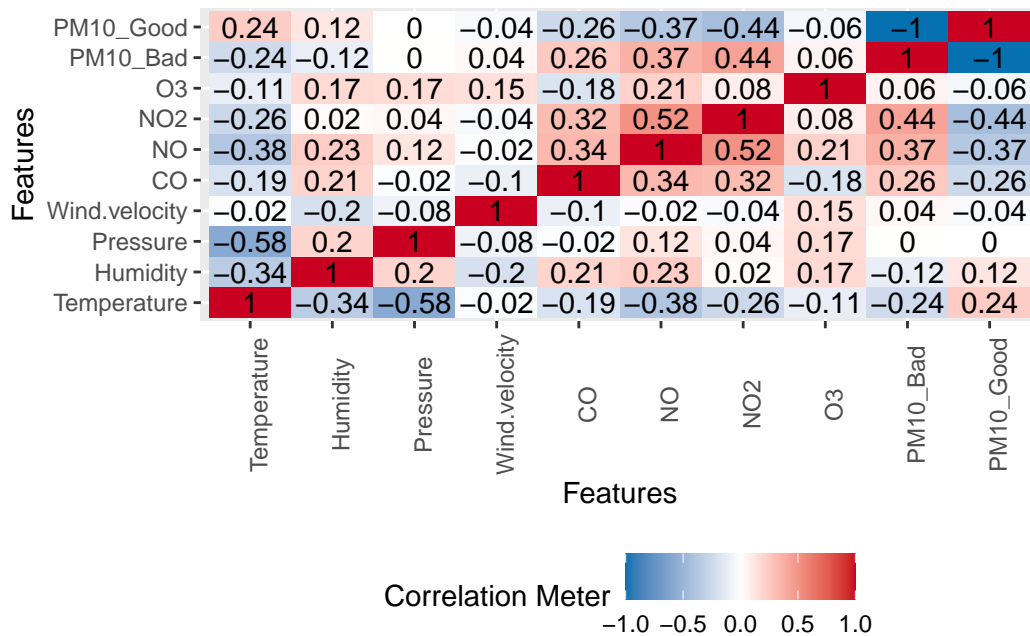
- The variable Particulate Matter (PM10) will be discretized using a threshold value of 45 $\mu\text{g}/\text{m}^3$, below which it will be categorized as “Good.” Above 45 $\mu\text{g}/\text{m}^3$, it will be assigned the value “Bad.”

```
y_col_name <- colnames(data)[10]
y_cut <- cut(data$PM10,breaks=c(-10,45,400),labels = c("Good","Bad"))
data$PM10 <- y_cut
```

EDA

Pearson correlation

```
plot_correlation(data)
```

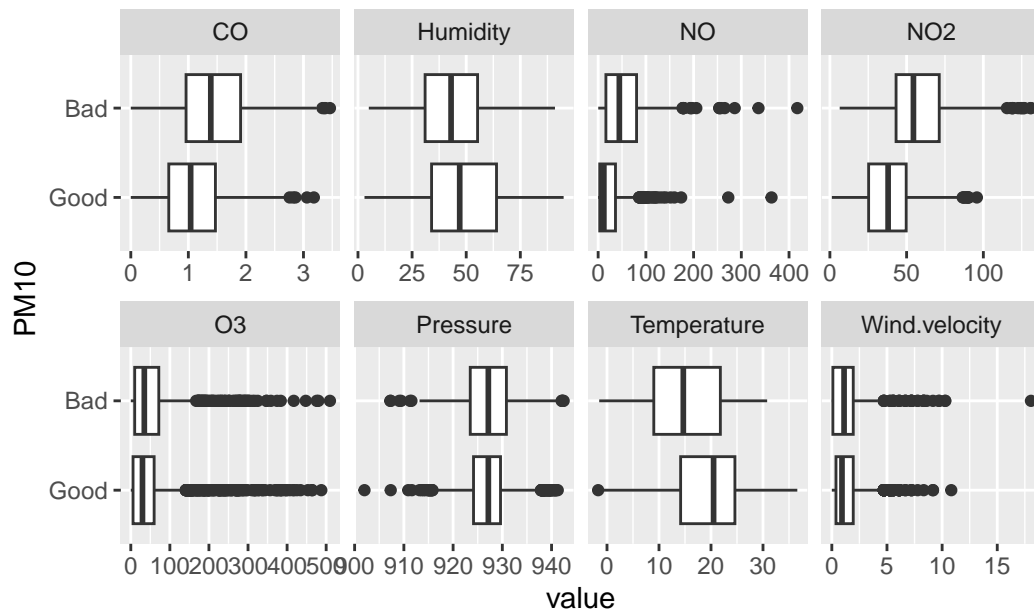



```
ggsave(filename = "./figs-to-paper/01-pearson-correlation.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

Box plot

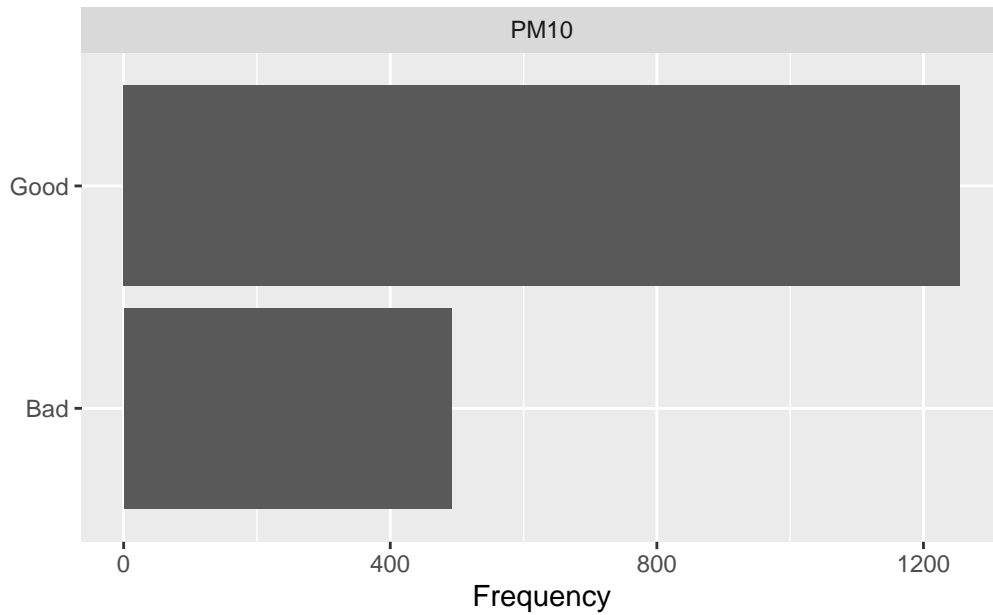
```
plot_boxplot(data, by = "PM10")
```



```
ggsave(filename = "./figs-to-paper/02-boxplot.tiff",units = "px", dpi=300)
```

Saving 1650 x 1050 px image

```
plot_bar(data)
```



```
ggsave(filename = "./figs-to-paper/03-dataset-desbalanceado.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

Machine learning models

Split train & test set

We split the dataset to use a 75 % for training and a 25 % for testing.

```
set.seed(123)
splits      <- initial_split(data, strata = PM10, prop = 3/4)

data_train <- training(splits) # 75 % train set
data_test  <- testing(splits)  # 25 % testing set
```

Binary clasification with Logistic regression

- We train two models to develop the binary classifier: logistic regression and random forest.

- We will use the tidymodels library.
- The logistic regression model is implemented in the glmnet package.
- mixture = 1 means L1 regularization (a pure Lasso model) will be used. A mixture value of 1 means that the glmnet model will potentially eliminate irrelevant predictors and choose a simpler model.
- penalty: This hyperparameter represents how much of this regularization we will use. We will adjust it during training to find the best value for making predictions with our data.

```
lr_mod <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet") # we use the package glmnet
```

Tidymodels recipe

- data_train dataset will be used to train the logistic model. We want to predict the variable PM10.
- step_normalize() creates a specification of a recipe step that will normalize numeric data to have a standard deviation of one and a mean of zero.

```
lr_recipe <-
  recipe(PM10 ~ ., data = data_train) %>%
  step_normalize(all_predictors())
```

Tidymodels workflow

```
lr_workflow <-
  workflow() %>% # create a workflow
  add_model(lr_mod) %>% # add the model
  add_recipe(lr_recipe) # add the recipe
```

Grid tuning

We have a hyperparameter to adjust: the penalty for L1 regularization. We can configure the grid manually using a one-column table with 30 candidate values.

```
lr_reg_grid <- tibble(penalty = 10^seq(-4, -1, length.out = 30))
```

List of the different values for the penalty hyperparameter to try in the training phase.

```
lr_reg_grid
```

```
# A tibble: 30 x 1
  penalty
  <dbl>
1 0.0001
2 0.000127
3 0.000161
4 0.000204
5 0.000259
6 0.000329
7 0.000418
8 0.000530
9 0.000672
10 0.000853
# i 20 more rows
```

Validation set

With the `strata` argument, random sampling is performed within the variable PM10 (the stratification variable). This can help ensure that the new samples have proportions equivalent to those in the original dataset. In the case of a categorical variable like PM10, sampling is performed separately within each class.

A validation dataset is used to tune the penalty hyperparameter. Within the training dataset, 80% is kept for training and 20% for validation.

Within the training dataset, we use a portion of it as a validation set to train with the different penalty values (the grid tuning we will perform).

```
set.seed(234)
# 20 %
val_set <- validation_split(data_train,
                           strata = PM10,
                           prop = 0.80)
```

Warning: `validation_split()` was deprecated in rsample 1.2.0.
i Please use `initial_validation_split()` instead.

Training execution

- The following code block executes everything: the recipe or instructions saved in `lr_workflow` plus the hyperparameter tuning (`tune_grid`).
- `ROC_AUC` is the classifier evaluation metric.

```
lr_res <-  
  lr_workflow %>%  
  tune_grid(val_set,  
            grid = lr_reg_grid,  
            control = control_grid(save_pred = TRUE),  
            metrics = metric_set(roc_auc))
```

```
lr_res
```

```
# Tuning results  
# Validation Set Split (0.8/0.2) using stratification  
# A tibble: 1 x 5  
  splits          id      .metrics      .notes      .predictions  
  <list>        <chr>    <list>      <list>      <list>  
1 <split [1047/263]> validation <tibble [30 x 5]> <tibble [0 x 3]> <tibble>
```

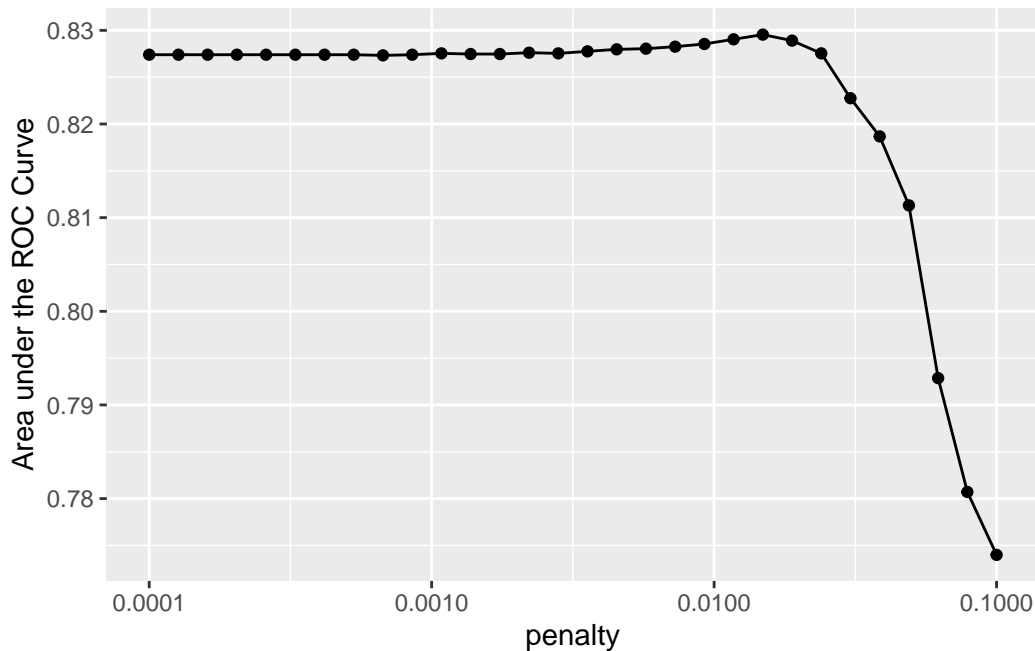
Hyperparameter tuning results

We plot the variation in ROC values for different penalty values.

The higher the ROC value, the better the models.

```
lr_plot <-  
  lr_res %>%  
  collect_metrics() %>%  
  ggplot(aes(x = penalty, y = mean)) +  
  geom_point() +  
  geom_line() +  
  ylab("Area under the ROC Curve") +  
  scale_x_log10(labels = scales::label_number())
```

```
lr_plot
```



```
ggsave(filename = "./figs-to-paper/04-log-reg-tunning-results.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

The graph above shows that model performance is generally better with lower penalty values, which suggests that most predictors are important to the model.

We also see a steep drop in the area under the ROC curve toward higher penalty values, which happens because a large enough penalty will remove all predictors from the model, and, as expected, predictive accuracy plummets with fewer predictors in the model.

Best Logistic Regression Models

We display the top 15 models based on the ROC metric using `show_best`. The higher the ROC value, the better the models.

The data is displayed in order from lowest to highest penalty value.

```
top_models <-
  lr_res %>%
  show_best(metric = "roc_auc", n = 15) %>%
  arrange(penalty)
top_models
```

```
# A tibble: 15 x 7
  penalty .metric .estimator mean    n std_err .config
  <dbl> <chr>   <chr>   <dbl> <int>  <dbl> <chr>
1 0.0001  roc_auc  binary   0.827     1     NA Preprocessor1_Model01
2 0.00108 roc_auc  binary   0.828     1     NA Preprocessor1_Model11
3 0.00137 roc_auc  binary   0.827     1     NA Preprocessor1_Model12
4 0.00174 roc_auc  binary   0.827     1     NA Preprocessor1_Model13
5 0.00221 roc_auc  binary   0.828     1     NA Preprocessor1_Model14
6 0.00281 roc_auc  binary   0.828     1     NA Preprocessor1_Model15
7 0.00356 roc_auc  binary   0.828     1     NA Preprocessor1_Model16
8 0.00452 roc_auc  binary   0.828     1     NA Preprocessor1_Model17
9 0.00574 roc_auc  binary   0.828     1     NA Preprocessor1_Model18
10 0.00728 roc_auc  binary   0.828     1     NA Preprocessor1_Model19
11 0.00924 roc_auc  binary   0.829     1     NA Preprocessor1_Model20
12 0.0117  roc_auc  binary   0.829     1     NA Preprocessor1_Model21
13 0.0149  roc_auc  binary   0.830     1     NA Preprocessor1_Model22
14 0.0189  roc_auc  binary   0.829     1     NA Preprocessor1_Model23
15 0.0240  roc_auc  binary   0.828     1     NA Preprocessor1_Model24
```

The same information as above, but this time sorted by descending ROC_AUC value.

The data is displayed sorted from lowest to highest according to the penalty value.

```
top_models %>%
  arrange(desc(mean))
```

```
# A tibble: 15 x 7
  penalty .metric .estimator mean    n std_err .config
  <dbl> <chr>   <chr>   <dbl> <int>  <dbl> <chr>
1 0.0149  roc_auc  binary   0.830     1     NA Preprocessor1_Model22
2 0.0117  roc_auc  binary   0.829     1     NA Preprocessor1_Model21
3 0.0189  roc_auc  binary   0.829     1     NA Preprocessor1_Model23
4 0.00924 roc_auc  binary   0.829     1     NA Preprocessor1_Model20
5 0.00728 roc_auc  binary   0.828     1     NA Preprocessor1_Model19
6 0.00574 roc_auc  binary   0.828     1     NA Preprocessor1_Model18
7 0.00452 roc_auc  binary   0.828     1     NA Preprocessor1_Model17
8 0.00356 roc_auc  binary   0.828     1     NA Preprocessor1_Model16
9 0.00221 roc_auc  binary   0.828     1     NA Preprocessor1_Model14
10 0.0240  roc_auc  binary   0.828     1     NA Preprocessor1_Model24
11 0.00108 roc_auc  binary   0.828     1     NA Preprocessor1_Model11
12 0.00281 roc_auc  binary   0.828     1     NA Preprocessor1_Model15
13 0.00137 roc_auc  binary   0.827     1     NA Preprocessor1_Model12
```



```
14 0.00174 roc_auc binary 0.827 1 NA Preprocessor1_Model13
15 0.0001 roc_auc binary 0.827 1 NA Preprocessor1_Model01
```

```
penalty_value <- lr_res %>%
  select_best(metric = "roc_auc") %>% # find the best hyperparameter combination given a per
  select(penalty)
```

The best logistic regression model is the one with $\text{penalty} = 0.0148735210729351$.

We observed minimal ROC variation at the other penalty values.

```
lr_best <- lr_res %>%
  collect_metrics() %>%
  arrange(desc(mean)) %>%
  slice(1)
lr_best
```

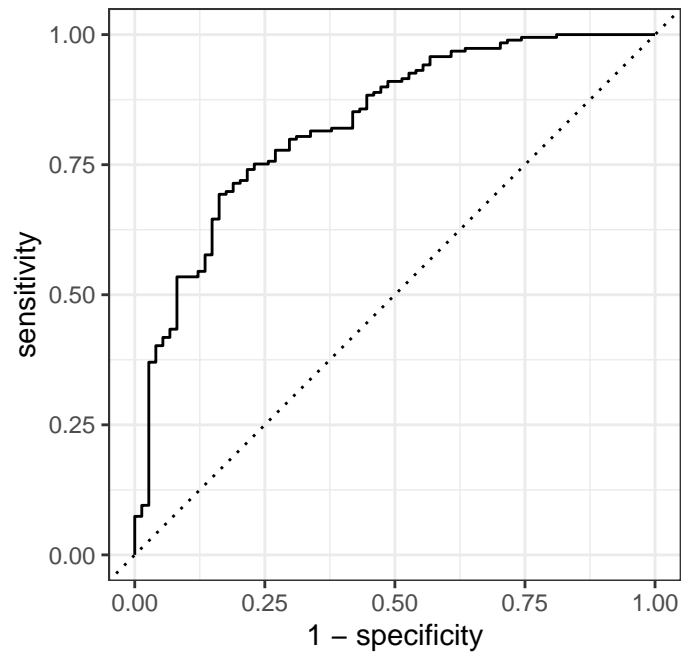
A tibble: 1 x 7

	penalty	.metric	.estimator	mean	n	std_err	.config
	<dbl>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
1	0.0149	roc_auc	binary	0.830	1	NA	Preprocessor1_Model22

Roc curve in training set

```
lr_auc <-
  lr_res %>%
  collect_predictions(parameters = lr_best) %>%
  roc_curve(PM10, .pred_Good) %>%
  mutate(model = "Logistic Regression")

autoplot(lr_auc)
```



```
ggsave(filename = "./figs-to-paper/05-log-reg-ROC-best-on-training.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

Results (test set)

Get the best model.

```
best_model <- lr_res %>%
  select_best(metric = "roc_auc")
```

```
best_model
```

```
# A tibble: 1 x 2
  penalty .config
  <dbl> <chr>
1  0.0149 Preprocessor1_Model22
```

Update the workflow.

```
final_wf <- lr_workflow %>%
  finalize_workflow(best_model)

final_wf
```

```
== Workflow =====
Preprocessor: Recipe
Model: logistic_reg()

-- Preprocessor -----
1 Recipe Step

* step_normalize()

-- Model -----
Logistic Regression Model Specification (classification)

Main Arguments:
  penalty = 0.0148735210729351
  mixture = 1

Computational engine: glmnet
```

We can use the `last_fit()` function with our finalized model; this function fits the finalized model on the full training dataset and evaluates the finalized model on the test data.

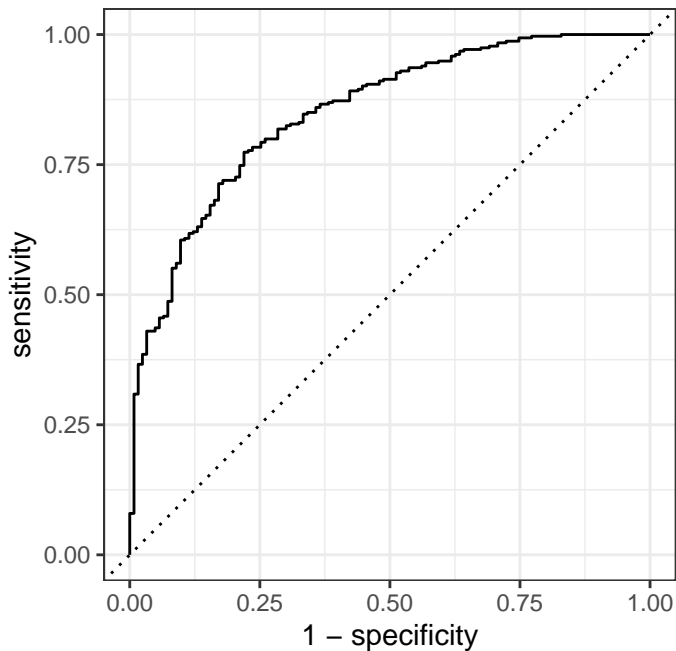
```
final_fit <-
  final_wf %>%
  last_fit(splits)
```

```
final_fit %>%
  collect_metrics()
```

```
# A tibble: 3 x 4
  .metric      .estimator .estimate .config
  <chr>        <chr>         <dbl> <chr>
1 accuracy    binary         0.801 Preprocessor1_Model1
2 roc_auc     binary         0.850 Preprocessor1_Model1
3 brier_class binary         0.136 Preprocessor1_Model1
```

ROC CURVE IN TEST SET

```
final_fit %>%  
  collect_predictions() %>%  
  roc_curve(PM10, .pred_Good) %>%  
  mutate(model = "Logistic Regression") %>%  
  autoplot()
```



```
ggsave(filename = "./figs-to-paper/06-log-reg-ROC-best-on-testing.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

We use the best LR model to predict on the test set.

```
lr_model <- extract_workflow(final_fit)  
  
# Class prediction  
pred_class <- predict(lr_model,  
  new_data = data_test,  
  type = "class")
```

```
# Prediction Probabilities
pred_proba <- predict(lr_model,
                      new_data = data_test,
                      type = "prob")
```

```
lr_results <- data_test %>%
  select(PM10) %>%
  bind_cols(pred_class, pred_proba)
```

Confusion matrix (on test set)

```
conf_mat(lr_results, truth = PM10,
         estimate = .pred_class)
```

	Truth	
Prediction	Good	Bad
Good	297	70
Bad	17	53

Accuracy, sensitivity, specificity, etc

```
summary(conf_mat(lr_results, truth = PM10,
                 estimate = .pred_class))
```

```
# A tibble: 13 x 3
```

	.metric	.estimator	.estimate
	<chr>	<chr>	<dbl>
1	accuracy	binary	0.801
2	kap	binary	0.434
3	sens	binary	0.946
4	spec	binary	0.431
5	ppv	binary	0.809
6	npv	binary	0.757
7	mcc	binary	0.462
8	j_index	binary	0.377
9	bal_accuracy	binary	0.688
10	detection_prevalence	binary	0.840

11 precision	binary	0.809
12 recall	binary	0.946
13 f_meas	binary	0.872

F-measure

```
f_meas(lr_results, truth = PM10,
       estimate = .pred_class)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>    <chr>        <dbl>
1 f_meas  binary          0.872
```

Random Forest

We train a random forest model for binary classification.

We use the random forest implementation from the ranger package.

We tune two hyperparameters: mtry and min_n, in training time.

We set the parameter trees to 100.

```
# detect the number of cores from the CPU
cores <- parallel::detectCores()

rf_mod <-
  rand_forest(mtry = tune(), min_n = tune(), trees = 100) %>%
  set_engine("ranger", num.threads = cores) %>% # use the ranger package
  set_mode("classification") # classification task
```

Tidymodels recipe

- data_train dataset will be used to train the RF model.
- We want to predict the variable PM10.

```
rf_recipe <-
  recipe(PM10 ~ ., data = data_train)
```

Tidymodels workflow setup

```
rf_workflow <-  
  workflow() %>%  
  add_model(rf_mod) %>% ## add model RF  
  add_recipe(rf_recipe) ## add recipe
```

RF training

Within the training dataset, we use a portion of it as a validation set to tune the hyperparameters (mtry and min_n).

```
set.seed(345)  
rf_res <-  
  rf_workflow %>%  
  tune_grid(val_set,  
    grid = 25, # grid size  
    control = control_grid(save_pred = TRUE),  
    metrics = metric_set(roc_auc))
```

i Creating pre-processing data to finalize unknown parameter: mtry

RF hyperparameter tuning results

Results for each value of mtry and min_n:

```
rf_res %>% collect_metrics()
```

A tibble: 25 x 8

	mtry	min_n	.metric	.estimator	mean	n	std_err	.config
	<int>	<int>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
1	1	24	roc_auc	binary	0.916	1	NA	Preprocessor1_Model01
2	1	13	roc_auc	binary	0.916	1	NA	Preprocessor1_Model02
3	1	33	roc_auc	binary	0.910	1	NA	Preprocessor1_Model03
4	1	5	roc_auc	binary	0.914	1	NA	Preprocessor1_Model04
5	2	19	roc_auc	binary	0.922	1	NA	Preprocessor1_Model05
6	2	27	roc_auc	binary	0.917	1	NA	Preprocessor1_Model06
7	2	36	roc_auc	binary	0.910	1	NA	Preprocessor1_Model07
8	3	11	roc_auc	binary	0.909	1	NA	Preprocessor1_Model08

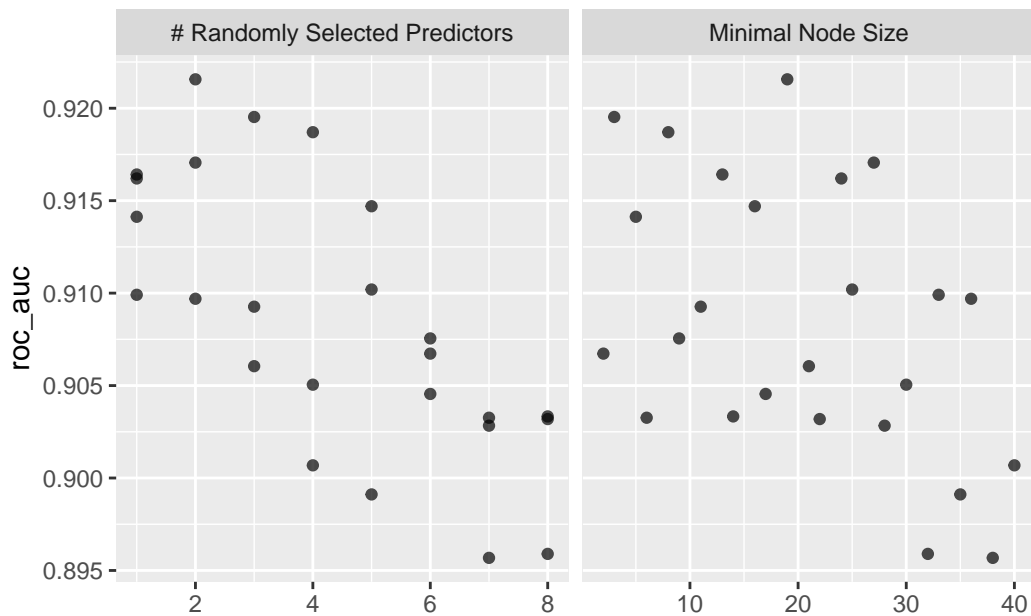
```

 9      3      3 roc_auc binary    0.920    1      NA Preprocessor1_Model09
10      3     21 roc_auc binary    0.906    1      NA Preprocessor1_Model10
# i 15 more rows

```

Plot hyperparameter tuning results.

```
autoplot(rf_res)
```



```
ggsave(filename = "./figs-to-paper/07-RF-tunning.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

RF best models

List the best RF models.

```

rf_res %>%
  show_best(metric = "roc_auc")

```



```
# A tibble: 5 x 8
  mtry min_n .metric .estimator mean      n std_err .config
  <int> <int> <chr>    <chr>    <dbl> <int>   <dbl> <chr>
1     2     19 roc_auc binary    0.922     1     NA Preprocessor1_Model05
2     3      3 roc_auc binary    0.920     1     NA Preprocessor1_Model09
3     4      8 roc_auc binary    0.919     1     NA Preprocessor1_Model13
4     2     27 roc_auc binary    0.917     1     NA Preprocessor1_Model06
5     1     13 roc_auc binary    0.916     1     NA Preprocessor1_Model02
```

We show the best one.

```
rf_best <-
  rf_res %>%
  select_best(metric = "roc_auc")
rf_best
```

```
# A tibble: 1 x 3
  mtry min_n .config
  <int> <int> <chr>
1     2     19 Preprocessor1_Model05
```

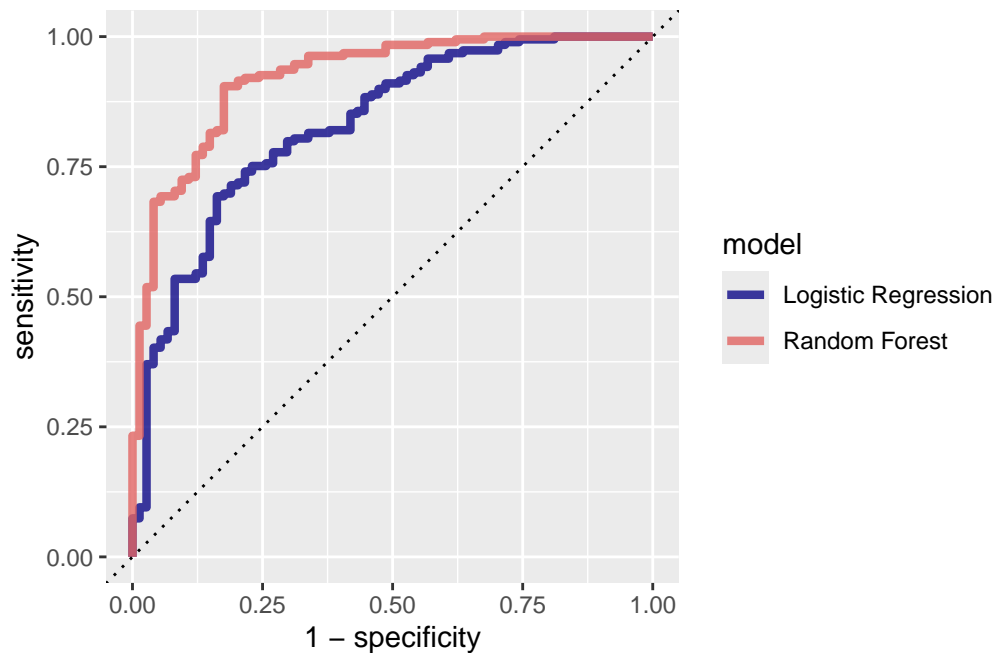
RF ROC curve dataset generation

```
# Para filtrar las predicciones solo para nuestro mejor modelo, podemos usar el argumento de

rf_auc <-
  rf_res %>%
  collect_predictions(parameters = rf_best) %>%
  roc_curve(PM10, .pred_Good) %>%
  mutate(model = "Random Forest")
```

ROC curve to compare Logistic regression model vs Random Forest model .

```
bind_rows(rf_auc, lr_auc) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```



```
ggsave(filename = "./figs-to-paper/08-ROC-on-training-set-by-model.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

Conclusion: the RF model was superior across the entire event probability threshold.

Final results

We built a model with the selected parameters, trained it, then predict using the test set.

```
# the last model
last_rf_mod <-
  rand_forest(mtry = rf_best$mtry, min_n = rf_best$min_n, trees = 100) %>%
  set_engine("ranger", num.threads = cores, importance = "impurity") %>%
  set_mode("classification")

# the last workflow
last_rf_workflow <-
  rf_workflow %>%
  update_model(last_rf_mod)
```

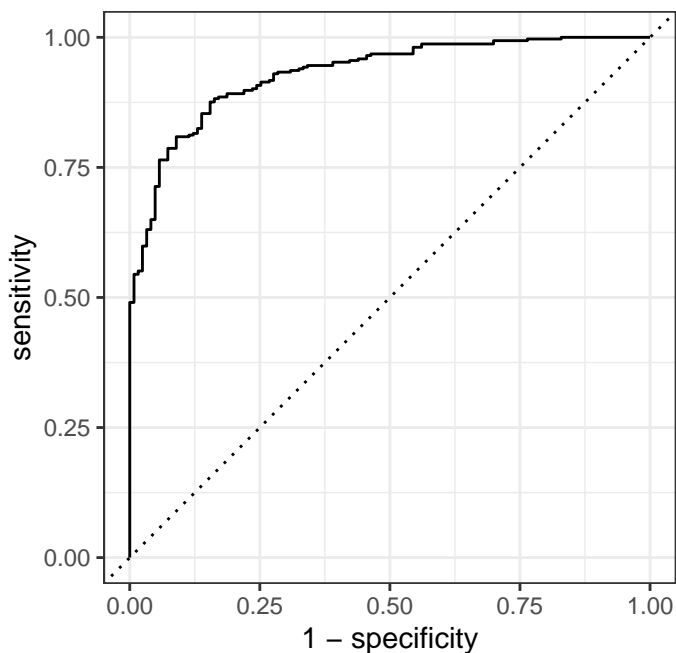
```
# the last fit
set.seed(345)
last_rf_fit <-
  last_rf_workflow %>%
  last_fit(splits)

last_rf_fit
```

```
# Resampling results
# Manual resampling
# A tibble: 1 x 6
  splits          id          .metrics .notes   .predictions .workflow
  <list>         <chr>         <list>  <list>   <list>       <list>
1 <split [1310/437]> train/test split <tibble> <tibble> <tibble>   <workflow>
```

ROC curve plot for Random Forest (test set)

```
last_rf_fit %>%
  collect_predictions() %>%
  roc_curve(PM10, .pred_Good) %>%
  autoplot()
```



```
ggsave(filename = "./figs-to-paper/09-ROC-RF-testset.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

Random Forest confusion matrix (test set)

```
RF_model <- extract_workflow(last_rf_fit)

# Class prediction
pred_class_rf <- predict(RF_model,
                        new_data = data_test,
                        type = "class")

# Prediction Probabilities
pred_proba_rf <- predict(RF_model,
                        new_data = data_test,
                        type = "prob")
```

```
RF_results <- data_test %>%
  select(PM10) %>%
  bind_cols(pred_class_rf, pred_proba_rf)
```

Print the confusion matrix.

```
conf_mat(RF_results, truth = PM10,
         estimate = .pred_class)
```

	Truth	
Prediction	Good	Bad
Good	294	39
Bad	20	84

Summary of metrics

```
summary(conf_mat(RF_results, truth = PM10,
                 estimate = .pred_class))
```

```
# A tibble: 13 x 3
```

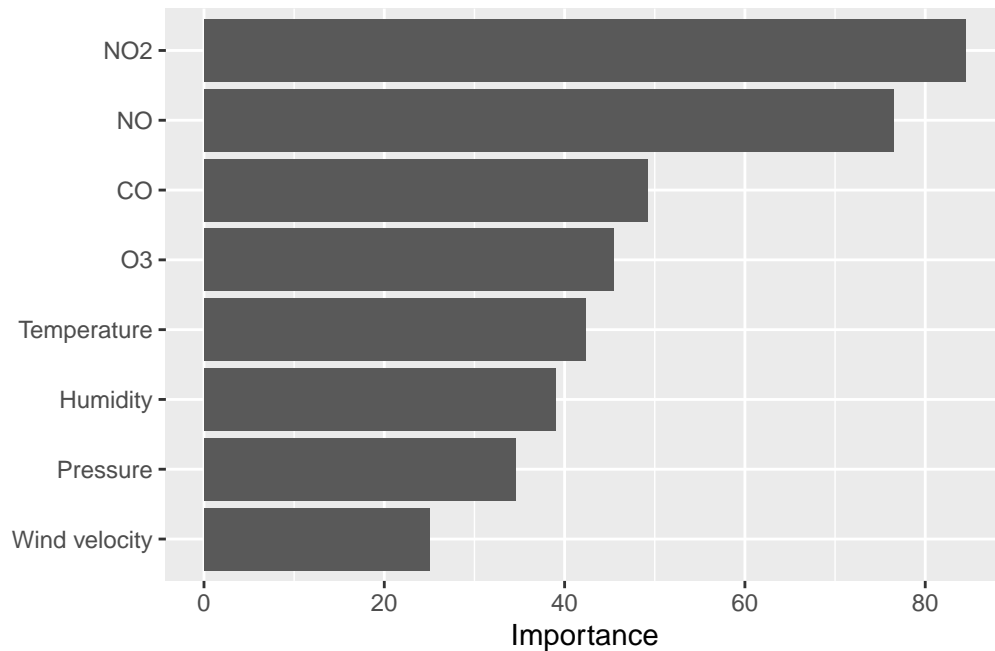
	.metric <chr>	.estimator <chr>	.estimate <dbl>
1	accuracy	binary	0.865
2	kap	binary	0.650
3	sens	binary	0.936
4	spec	binary	0.683
5	ppv	binary	0.883
6	npv	binary	0.808
7	mcc	binary	0.654
8	j_index	binary	0.619
9	bal_accuracy	binary	0.810
10	detection_prevalence	binary	0.762
11	precision	binary	0.883
12	recall	binary	0.936
13	f_meas	binary	0.909

Variable importance Score

Compute the VIP.

When using the `vip` function with a random forest model, the default method computes the mean decrease in impurity (or Gini importance) for each variable. This is calculated by accumulating the improvement in the split criterion at each split in each tree, and normalizing by the standard deviation of the differences.

```
last_rf_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10) # check vip package documentation
```



```
ggsave(filename = "./figs-to-paper/10-RF-VIP.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

SHAP values

Recordamos que `rf_best` no es un objeto del modelo entrenado, sino un conjunto de hiperparámetros seleccionados que representan el mejor modelo. Requiere un objeto de la librería `ranger` que represente el mejor modelo random forest, para luego ser usado para SHAP

Setup and computing

```
rf_best <- rf_res %>% select_best(metric = "roc_auc")
# hiperparámetros "ganadores"
rf_best
```

```
# A tibble: 1 x 3
  mtry min_n .config
<int> <int> <chr>
1     2    19 Preprocessor1_Model105
```

```
# workflow setup: update the hyperparameters
final_rf_workflow <- rf_workflow %>% finalize_workflow(rf_best)

# Train a RF model
rf_final_fit <- final_rf_workflow %>% fit(data = data_train)

# Get the final model (a ranger::ranger object)
modelo_final <- extract_fit_parsnip(rf_final_fit)$fit
modelo_final
```

Ranger result

Call:

```
ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~2L, x), num.trees = ~1
```

```
Type: Probability estimation
Number of trees: 100
Sample size: 1310
Number of independent variables: 8
Mtry: 2
Target node size: 19
Variable importance mode: none
Splitrule: gini
OOB prediction error (Brier s.): 0.1016491
```

modelo_final is an object of the class ranger

```
class(modelo_final)
```

```
[1] "ranger"
```

We create a dataset called X with all the predictors features (without PM10)

```
library(fastshap)
```

Adjuntando el paquete: 'fastshap'

The following object is masked from 'package:vip':

gen_friedman

The following object is masked from 'package:dplyr':

explain

```
library(shapviz)
```

```
# Creo X (sin PM10, solo predictoras)
```

```
X <- data %>% select(-PM10)
```

```
head(X)
```

```
# A tibble: 6 x 8
```

	Temperature	Humidity	Pressure	`Wind velocity`	CO	NO	NO2	O3
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	53	938.	0	2.52	99.8	65.6	9.45
2	6.9	42	938.	1.11	1.97	94.2	77.0	13.7
3	10.1	31	938.	1.11	1.91	33.0	48.0	22.0
4	10.4	32	938.	1.11	1.56	15.1	31.4	56.3
5	11.6	30	937.	2.5	1.39	15.6	38.2	146.
6	12.4	30	936.	2.5	1.23	10.2	34.7	290.

Prediction function definition

```
#
```

```
pred_fun <- function(object, newdata) {  
  # get the probabilities matrix  
  prob_matrix <- predict(object, data = newdata, response = "prob")  
  return(prob_matrix$predictions[,1]) # return predictions for class Good  
  # return(prob_matrix$predictions[,2]) # return predictions for class Bad  
}
```

```
# pred_fun(modelo_final, X)$predictions
```

Calculate SHAP values with fastshap package


```
library(kernelshap)
library(shapviz)

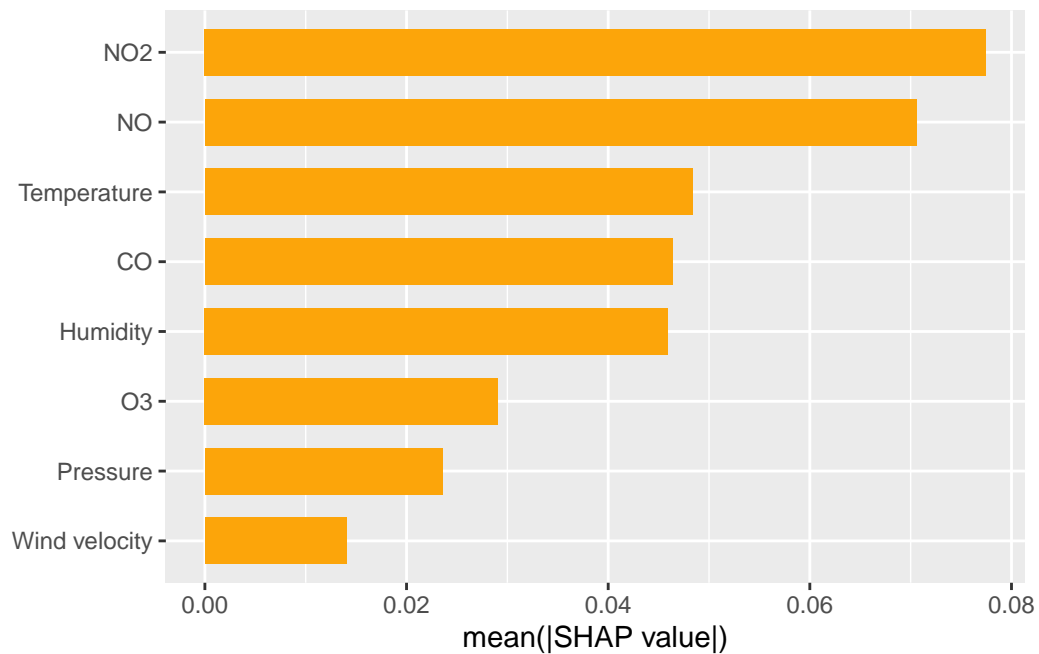
# Cálculo de SHAP values con fastshap
shap_values <- fastshap::explain(
  object = modelo_final,
  X = X,
  pred_wrapper = pred_fun,
  nsim = 100, # Aumentar para mayor precisión
)
```

SHAP values visualization

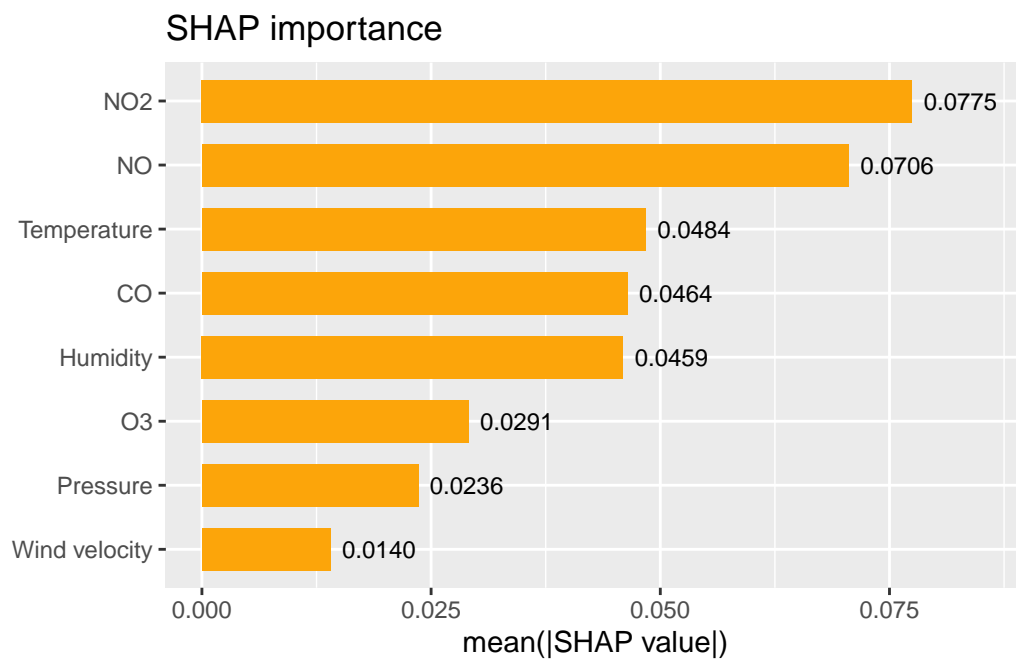
Importance plot.

```
shap <- shapviz(shap_values, X = X )

# Gráfico de importancia
sv_importance(shap)
```



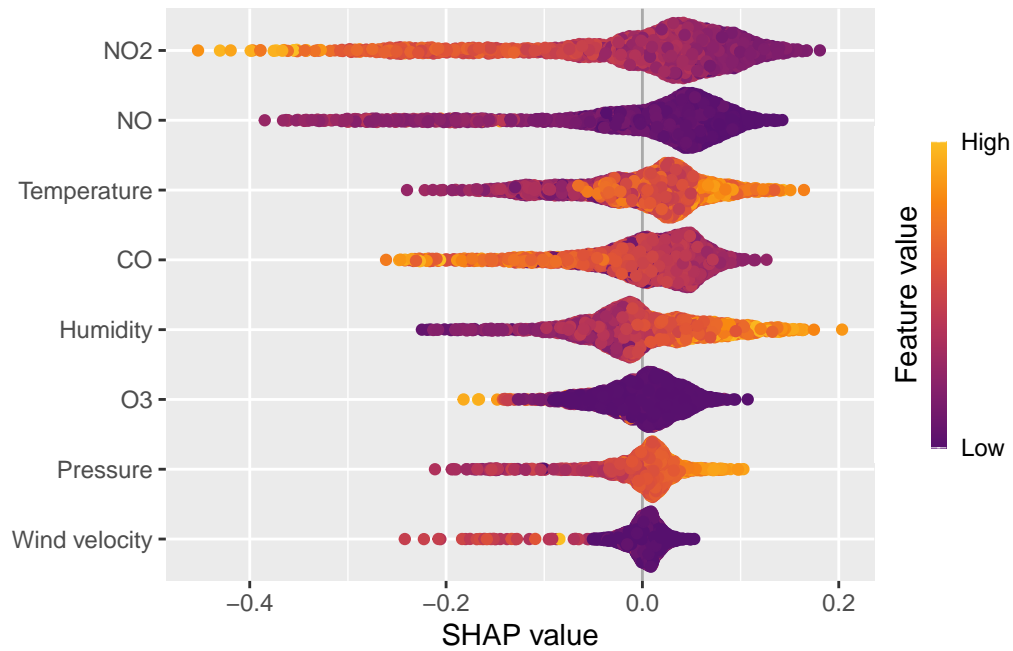
```
sv_importance(shap,show_numbers = TRUE) +
  ggtitle("SHAP importance")
```



```
ggsave(filename = "./figs-to-paper/12-SHAP_importance.tiff",units = "px", dpi=300)
```

Saving 1650 x 1050 px image

```
sv_importance(shap,"bee")
```



```
ggsave(filename = "./figs-to-paper/13-SHAP_importance_bee.tiff", units = "px", dpi=300)
```

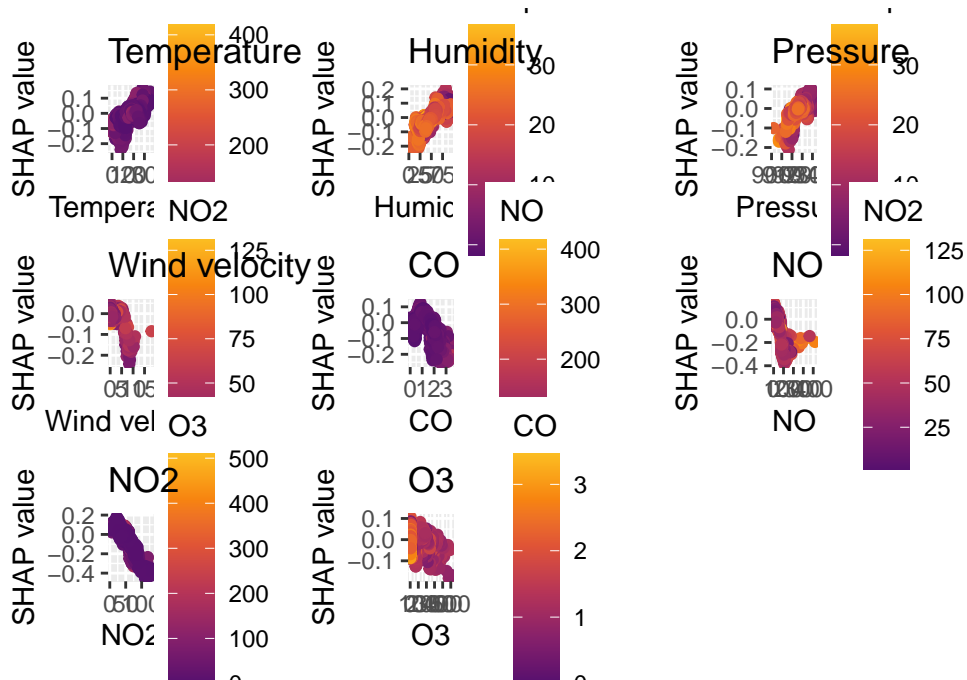
Saving 1650 x 1050 px image

Plot interaction

```
colnames(data)[-9]
```

```
[1] "Temperature" "Humidity"     "Pressure"     "Wind velocity"
[5] "CO"          "NO"          "NO2"         "O3"
```

```
sv_dependence(shap, colnames(data)[-9])
```



```
ggsave(filename = "./figs-to-paper/14-SHAP_dependence.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

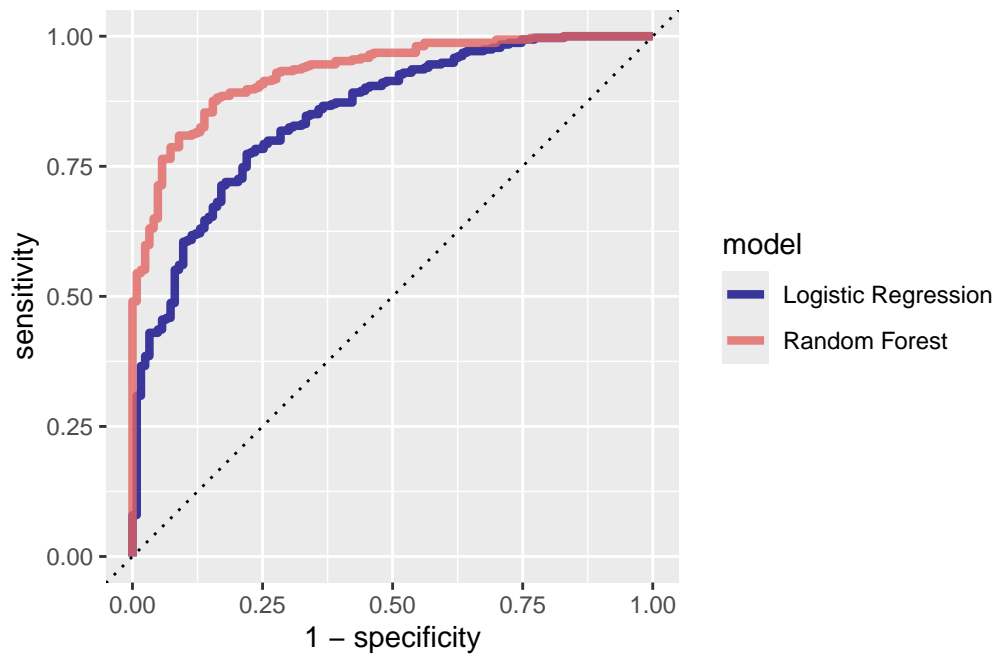
FINAL RESULTS

ROC CURVE

```
lr_auc_f <- final_fit %>%
  collect_predictions() %>%
  roc_curve(PM10, .pred_Good) %>%
  mutate(model = "Logistic Regression")
```

```
rf_auc_f <- last_rf_fit %>%
  collect_predictions() %>%
  roc_curve(PM10, .pred_Good) %>%
  mutate(model = "Random Forest")
```

```
bind_rows(rf_auc_f, lr_auc_f) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model)) +
  geom_path(lwd = 1.5, alpha = 0.8) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option = "plasma", end = .6)
```



```
ggsave(filename = "./figs-to-paper/11-ROC-on-test-set.tiff", units = "px", dpi=300)
```

Saving 1650 x 1050 px image

Metrics

Agrego una fila con el resultado ROC_AUC

```
roc_auc(lr_results, truth = PM10, .pred_Good)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>    <chr>        <dbl>
1 roc_auc binary      0.850
```

```
m_lr <- summary(conf_mat(lr_results, truth = PM10,
  estimate = .pred_class)) %>%
  bind_rows(roc_auc(lr_results, truth = PM10, .pred_Good)) %>%
  mutate(model = "Logistic Regression")
```

```
m_lr
```

```
# A tibble: 14 x 4
```

	.metric	.estimator	.estimate	model
	<chr>	<chr>	<dbl>	<chr>
1	accuracy	binary	0.801	Logistic Regression
2	kap	binary	0.434	Logistic Regression
3	sens	binary	0.946	Logistic Regression
4	spec	binary	0.431	Logistic Regression
5	ppv	binary	0.809	Logistic Regression
6	npv	binary	0.757	Logistic Regression
7	mcc	binary	0.462	Logistic Regression
8	j_index	binary	0.377	Logistic Regression
9	bal_accuracy	binary	0.688	Logistic Regression
10	detection_prevalence	binary	0.840	Logistic Regression
11	precision	binary	0.809	Logistic Regression
12	recall	binary	0.946	Logistic Regression
13	f_meas	binary	0.872	Logistic Regression
14	roc_auc	binary	0.850	Logistic Regression

```
roc_auc(RF_results, truth = PM10, .pred_Good)
```

```
# A tibble: 1 x 3
```

	.metric	.estimator	.estimate
	<chr>	<chr>	<dbl>
1	roc_auc	binary	0.930

```
m_rf <- summary(conf_mat(RF_results, truth = PM10, estimate = .pred_class)) %>%
  bind_rows(roc_auc(RF_results, truth = PM10, .pred_Good)) %>%
  mutate(model = "Random Forest")
```

```
m_rf
```

```
# A tibble: 14 x 4
```

	.metric	.estimator	.estimate	model
	<chr>	<chr>	<dbl>	<chr>
1	accuracy	binary	0.865	Random Forest
2	kap	binary	0.650	Random Forest
3	sens	binary	0.936	Random Forest
4	spec	binary	0.683	Random Forest
5	ppv	binary	0.883	Random Forest
6	npv	binary	0.808	Random Forest
7	mcc	binary	0.654	Random Forest
8	j_index	binary	0.619	Random Forest
9	bal_accuracy	binary	0.810	Random Forest
10	detection_prevalence	binary	0.762	Random Forest
11	precision	binary	0.883	Random Forest
12	recall	binary	0.936	Random Forest
13	f_meas	binary	0.909	Random Forest
14	roc_auc	binary	0.930	Random Forest

Results table to compare each model

```
bind_rows(m_rf, m_lr) %>%
  select(-one_of(c(".estimator"))) %>%
  pivot_wider( names_from = "model", values_from = ".estimate")
```

A tibble: 14 x 3

	.metric	`Random Forest`	`Logistic Regression`
	<chr>	<dbl>	<dbl>
1	accuracy	0.865	0.801
2	kap	0.650	0.434
3	sens	0.936	0.946
4	spec	0.683	0.431
5	ppv	0.883	0.809
6	npv	0.808	0.757
7	mcc	0.654	0.462
8	j_index	0.619	0.377
9	bal_accuracy	0.810	0.688
10	detection_prevalence	0.762	0.840
11	precision	0.883	0.809
12	recall	0.936	0.946
13	f_meas	0.909	0.872
14	roc_auc	0.930	0.850

Save the results in a csv file.

```
bind_rows(m_rf, m_lr) %>%
  select(-one_of(c(".estimator"))) %>%
  pivot_wider( names_from = "model", values_from = ".estimate") %>%
  write_csv(file="./figs-to-paper/final-results.csv")
```

R session info

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-linux-gnu
Running under: Ubuntu 24.10
```

```
Matrix products: default
```

```
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
```

```
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0
```

```
locale:
```

```
[1] LC_CTYPE=es_ES.UTF-8      LC_NUMERIC=C
[3] LC_TIME=es_ES.UTF-8      LC_COLLATE=es_ES.UTF-8
[5] LC_MONETARY=es_ES.UTF-8  LC_MESSAGES=es_ES.UTF-8
[7] LC_PAPER=es_ES.UTF-8     LC_NAME=C
[9] LC_ADDRESS=C             LC_TELEPHONE=C
[11] LC_MEASUREMENT=es_ES.UTF-8 LC_IDENTIFICATION=C
```

```
time zone: America/Argentina/Mendoza
```

```
tzcode source: system (glibc)
```

```
attached base packages:
```

```
[1] stats      graphics  grDevices datasets  utils      methods    base
```

```
other attached packages:
```

```
[1] fastshap_0.1.1  naniar_1.1.0      stringr_1.5.1     shapviz_0.9.7
[5] kernelshap_0.7.0 readxl_1.4.5      ranger_0.17.0     vip_0.4.1
[9] glmnet_4.1-8    Matrix_1.7-0      yardstick_1.3.2   workflowsets_1.1.0
[13] workflows_1.2.0 tune_1.3.0         tibble_3.2.1      rsample_1.3.0
[17] recipes_1.2.1   purrr_1.0.4       parsnip_1.3.1     modeldata_1.4.0
```


[21] infer_1.0.7	ggplot2_3.5.2	dials_1.4.0	scales_1.3.0
[25] broom_1.0.8	tidymodels_1.3.0	DataExplorer_0.8.3	tidyr_1.3.1
[29] dplyr_1.1.4	readr_2.1.5		

loaded via a namespace (and not attached):

[1] gridExtra_2.3	rlang_1.1.5	magrittr_2.0.3
[4] furrr_0.3.1	compiler_4.4.1	systemfonts_1.2.2
[7] reshape2_1.4.4	vctrs_0.6.5	lhs_1.2.0
[10] crayon_1.5.3	pkgconfig_2.0.3	shape_1.4.6.1
[13] fastmap_1.2.0	backports_1.5.0	labeling_0.4.3
[16] utf8_1.2.4	rmarkdown_2.29	prodlim_2024.06.25
[19] tzdb_0.5.0	ragg_1.4.0	visdat_0.6.0
[22] bit_4.6.0	xfun_0.52	jsonlite_2.0.0
[25] parallel_4.4.1	R6_2.6.1	stringi_1.8.7
[28] parallelly_1.43.0	rpart_4.1.23	lubridate_1.9.4
[31] cellranger_1.1.0	xgboost_1.7.9.1	Rcpp_1.0.14
[34] iterators_1.0.14	knitr_1.50	future.apply_1.11.3
[37] splines_4.4.1	nnet_7.3-19	igraph_2.1.4
[40] timechange_0.3.0	tidyselect_1.2.1	rstudioapi_0.17.1
[43] yaml_2.3.10	timeDate_4041.110	codetools_0.2-20
[46] listenv_0.9.1	plyr_1.8.9	lattice_0.22-6
[49] withr_3.0.2	evaluate_1.0.3	future_1.40.0
[52] survival_3.7-0	pillar_1.10.2	renv_1.1.4
[55] foreach_1.5.2	generics_0.1.3	vroom_1.6.5
[58] hms_1.1.3	munsell_0.5.1	globals_0.16.3
[61] class_7.3-22	glue_1.8.0	tools_4.4.1
[64] data.table_1.17.0	gower_1.0.2	grid_4.4.1
[67] ipred_0.9-15	colorspace_2.1-1	patchwork_1.3.0
[70] networkD3_0.4	sfd_0.1.0	cli_3.6.4
[73] DiceDesign_1.10	textshaping_1.0.0	viridisLite_0.4.2
[76] lava_1.8.1	gtable_0.3.6	GPfit_1.0-8
[79] digest_0.6.37	farver_2.1.2	htmlwidgets_1.6.4
[82] htmltools_0.5.8.1	lifecycle_1.0.4	hardhat_1.4.1
[85] bit64_4.6.0-1	sparsevctrs_0.3.2	MASS_7.3-61