

document

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 1.2.0 --

v broom	1.0.6	v recipes	1.0.10
v dials	1.2.1	v rsample	1.2.1
v ggplot2	3.5.1	v tibble	3.2.1
v infer	1.0.7	v tune	1.2.1
v modeldata	1.3.0	v workflows	1.1.4
v parsnip	1.2.1	v workflowsets	1.1.0
v purrr	1.0.2	v yardstick	1.3.1

```
-- 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()
* Search for functions across packages at https://www.tidymodels.org/find/
```

```
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)
```

Load dataset

```
database <- read_rds("../data/database.rds")
colnames(database)
```

```

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

```

```
nrow(database)
```

```
[1] 1105
```

```
summary(database)
```

Estación	Temperatura (°C)	Humedad (%)	Presión (hPa)
Length:1105	Min. : 0.00	Min. : 3.00	Min. :902.0
Class :character	1st Qu.:11.40	1st Qu.:32.00	1st Qu.:923.2
Mode :character	Median :18.00	Median :46.00	Median :927.3
	Mean :17.32	Mean :48.04	Mean :926.9
	3rd Qu.:23.50	3rd Qu.:62.00	3rd Qu.:930.5
	Max. :36.60	Max. :95.00	Max. :940.0

Velocidad de viento (m/s)	CO (mg/m3)	NO (ug/m3)	NO2 (ug/m3)
Min. : 0.000	Min. :0.000	Min. : 0.02	Min. : 1.28
1st Qu.: 0.310	1st Qu.:0.740	1st Qu.: 3.53	1st Qu.: 30.71
Median : 1.110	Median :1.120	Median : 26.00	Median : 45.56
Mean : 1.484	Mean :1.172	Mean : 35.36	Mean : 44.17
3rd Qu.: 1.940	3rd Qu.:1.580	3rd Qu.: 55.15	3rd Qu.: 54.11
Max. :18.060	Max. :3.460	Max. :363.27	Max. :131.05

NOX (ug/m3)	O3 (ug/m3)	PM10 (ug/m3)
Min. : 4.15	Min. : 0.00	Min. : 0.00
1st Qu.: 40.11	1st Qu.: 13.05	1st Qu.: 15.00
Median : 93.49	Median : 30.55	Median : 29.00
Mean : 98.40	Mean : 51.02	Mean : 36.75
3rd Qu.:137.60	3rd Qu.: 43.86	3rd Qu.: 49.00
Max. :559.91	Max. :487.52	Max. :318.00
		NA's :11

time
Min. :2022-02-11 17:00:00.00
1st Qu.:2022-05-02 10:00:00.00
Median :2023-01-13 02:00:00.00

```
Mean      :2022-11-29 21:23:43.71
3rd Qu.   :2023-03-27 15:00:00.00
Max.      :2023-10-17 07:00:00.00
```

Cleaning

- Tener en cuenta Temperatura, Humedad relativa, Presión atmosférica, Velocidad de viento, CO, NO, NO2, O3 como variables predictoras de PM10.
- Quitar la variable NOX
- Remuevo NA

```
data <- database %>% select(-one_of(c("Estación","time","NOX (ug/m3)")))  
data <- data[complete.cases(data),]
```

- El dataset a entrenar tiene 1094 muestras o filas.
- Cuenta con 9 variables o columnas.
- Las variables se llaman: Temperatura (°C), Humedad (%), Presión (hPa), Velocidad de viento (m/s), CO (mg/m3), NO (ug/m3), NO2 (ug/m3), O3 (ug/m3), PM10 (ug/m3).

```
colnames(data)
```

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

- Procedo a quitar las unidades al nombre de cada variable. Esto hará más legible los gráficos. Por supuesto que en paper debe explicarse cada variable y sus unidades.

```
library(stringr)
```

Adjuntando el paquete: 'stringr'

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

```
fixed
```

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

- Se lista un resumen de los datos.

```
summary(data)
```

Temperatura	Humedad	Presión	Velocidad de viento
Min. : 0.00	Min. : 5.0	Min. : 902.0	Min. : 0.000
1st Qu.: 11.31	1st Qu.: 32.0	1st Qu.: 923.3	1st Qu.: 0.310
Median : 17.90	Median : 46.0	Median : 927.3	Median : 1.110
Mean : 17.25	Mean : 48.2	Mean : 927.0	Mean : 1.483
3rd Qu.: 23.45	3rd Qu.: 62.0	3rd Qu.: 930.5	3rd Qu.: 1.940
Max. : 36.60	Max. : 95.0	Max. : 940.0	Max. : 18.060

CO	NO	NO2	O3
Min. : 0.000	Min. : 0.020	Min. : 1.28	Min. : 0.00
1st Qu.: 0.740	1st Qu.: 3.542	1st Qu.: 30.94	1st Qu.: 13.05
Median : 1.120	Median : 26.045	Median : 45.73	Median : 30.61
Mean : 1.174	Mean : 35.203	Mean : 44.38	Mean : 51.27
3rd Qu.: 1.577	3rd Qu.: 55.203	3rd Qu.: 54.13	3rd Qu.: 43.81
Max. : 3.460	Max. : 196.860	Max. : 131.05	Max. : 487.52

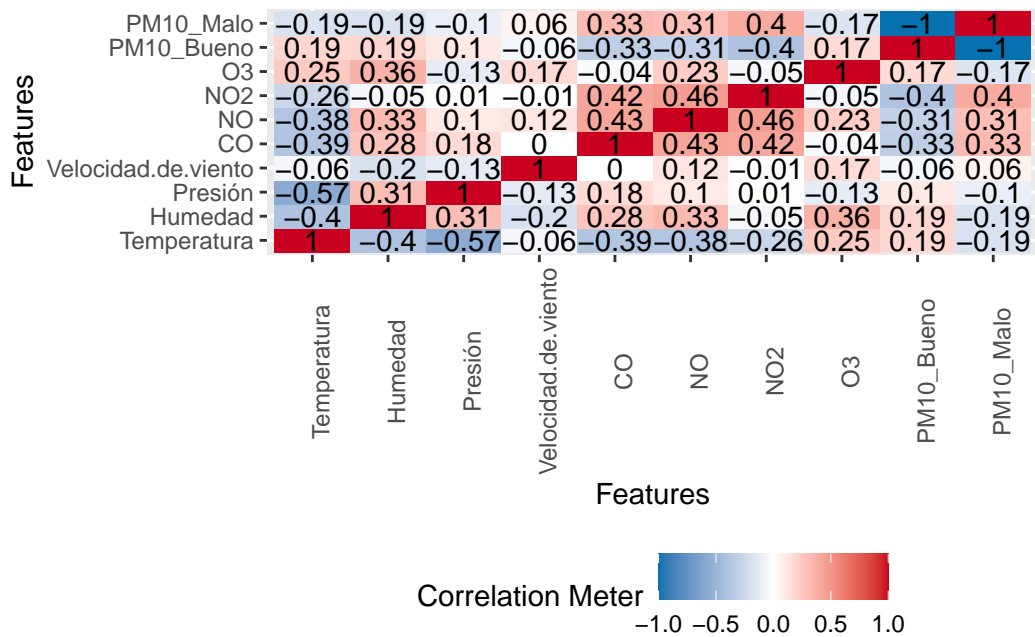
PM10
Min. : 0.00
1st Qu.: 15.00
Median : 29.00
Mean : 36.75
3rd Qu.: 49.00
Max. : 318.00

- Discretizar la variable Material Particulado (PM10) tomando como umbral el valor de 45 µg/m3, por debajo del cual se categorizará como “Bueno”. Por encima de 45 µg/m3, se asignará el valor “Malo”.

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

Pearson correlation

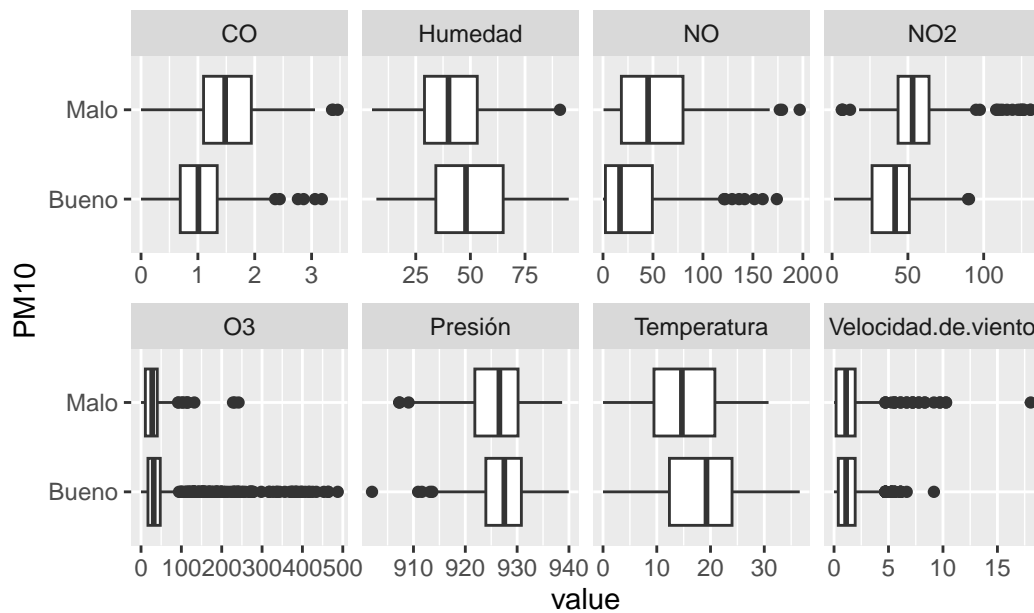
```
plot_correlation(data)
```



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

Saving 1650 x 1050 px image

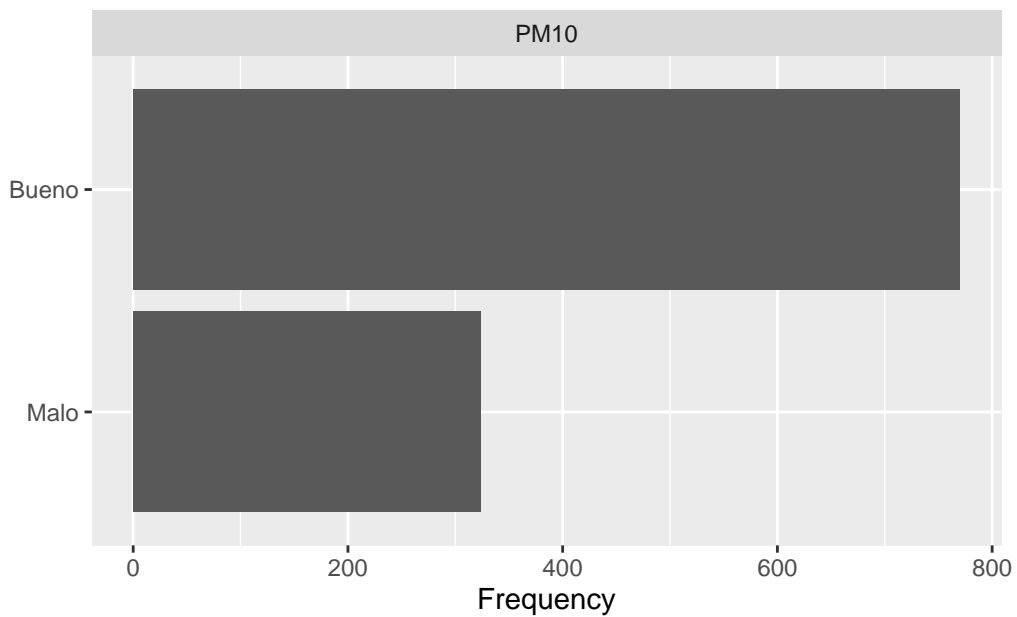
```
plot_boxplot(data, by = "PM10")
```



```
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

Modelos

Separación de sets de datos

Los datos del dataset data, se usa un 75 % o 3/4 partes para entrenamiento y un 25% para testeo.

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

data_train <- training(splits) # 75 % entrenamiento
data_test  <- testing(splits)  # 25 % en testeo
```

Clasificación binaria

Para desarrollar el clasificador binario entrenamos dos modelos: regresión logística (logistic regression) y random forest.

Usaremos la librería tidymodels.

Regresion logistica

- El modelo de regresión logística a usar está implementado en la librería glmnet.
- penalty: representa cuánta de esa regularización utilizaremos. Este es un hiperparámetro que ajustaremos durante el entrenamiento para encontrar el mejor valor para hacer predicciones con nuestros datos.
- mixture = 1 significa que se usará una regularización L1 (L1 regularization, pure lasso model), mixture en un valor de uno significa que el modelo glmnet eliminará potencialmente los predictores irrelevantes y elegirá un modelo más simple.

```
lr_mod <-
  logistic_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")
```


Receta.

- usamos los datos de entrenamiento (data_train) para predecir la 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())
```

Creamos el flujo de trabajo de tidymodels: el paso a paso de lo que queremos que ejecute.

```
lr_workflow <-  
  workflow() %>%  
  add_model(lr_mod) %>%  
  add_recipe(lr_recipe)
```

Grid tuning

Dado que solo tenemos un hiperparámetro para ajustar aquí, podemos configurar la cuadrícula manualmente usando un tibble de una columna con 30 valores candidatos:

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

Estos son los valores que se probarán en el entrenamiento: diferentes valores para el hiperparámetro penalty.

```
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
```

Con el argumento `strata`, el muestreo aleatorio (random sampling) se realiza dentro de la variable PM10 (the stratification variable). Esto puede ayudar a garantizar que las nuevas muestras tengan proporciones equivalentes a las del conjunto de datos original. En el caso de una variable categórica como PM10, el muestreo se realiza por separado dentro de cada clase.

Para el tuneo del hiperparámetro `penalty` se utiliza un set de datos de validación. Dentro del dataset de entrenamiento, un 80 % se mantiene para entrenar, y se usa el 20 % para validar.

Dentro del conjunto de datos de entrenamiento, usamos una porción del mismo como conjunto de validación para entrenar con los distintos valores de `penalty` (el grid tuning que realizaremos).

```
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.

En el siguiente bloque de código se ejecuta todo: la receta o indicaciones que quedaron guardadas en `lr_workflow` más el tuneo de hiperparámetros `tune_grid`.

Como métrica de evaluación del clasificador se utiliza `ROC_AUC`.

```
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
```

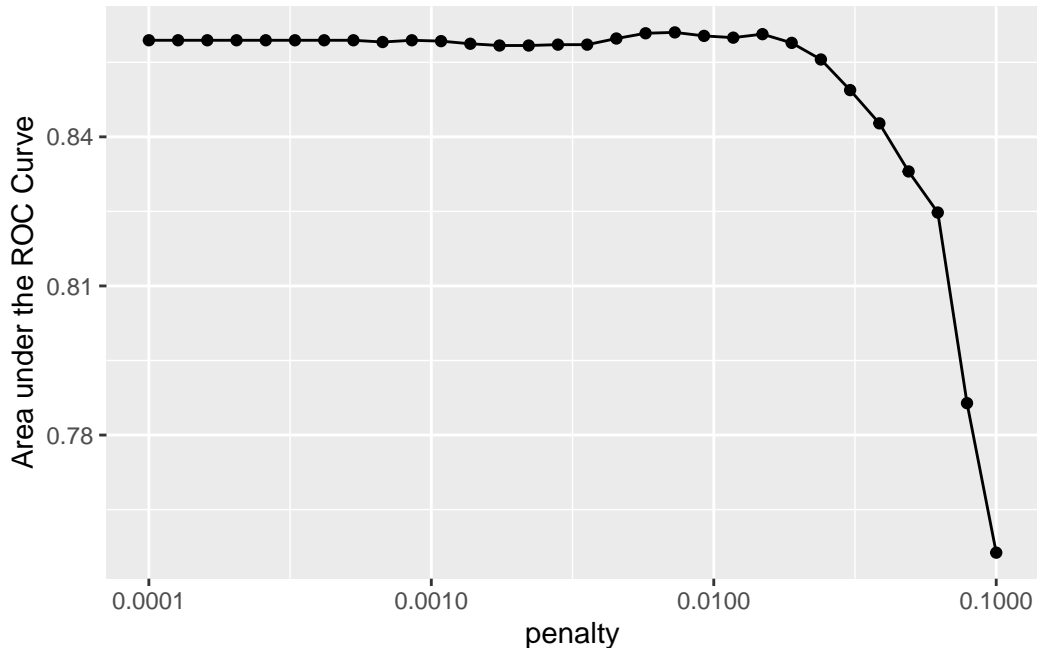
```
# Tuning results
# Validation Set Split (0.8/0.2) using stratification
# A tibble: 1 x 5
  splits          id      .metrics      .notes      .predictions
  <list>         <chr>    <list>      <list>      <list>
1 <split [655/165]> validation <tibble [30 x 5]> <tibble [0 x 3]> <tibble>
```

Resultados del tuneo de hiperparámetro

Graficamos la variación de los valores de ROC ante diferentes valores de penalty. Mientras más alto es el valor de ROC, mejores son dichos modelos.

```
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

Este gráfico nos muestra que el rendimiento del modelo es generalmente mejor con los valores de penalización más bajos. Esto sugiere que la mayoría de los predictores son importantes para el modelo. También vemos una caída pronunciada en el área bajo la curva ROC hacia los valores de penalización más altos. Esto sucede porque una penalización lo suficientemente grande eliminará todos los predictores del modelo y, como era de esperar, la precisión predictiva se desploma usando menos predictores en el modelo.

Mejores modelos de Logistic Regression

Mostramos los mejores 15 modelos según la métrica ROC usando `show_best`. Mientras más alto es el valor de ROC, mejores son dichos modelos.

Los datos se muestran ordenados de menor a mayor según el valor de `penalty`.

```
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	0.859	1	NA	Preprocessor1_Model01
2	0.000127	roc_auc	binary	0.859	1	NA	Preprocessor1_Model02
3	0.000161	roc_auc	binary	0.859	1	NA	Preprocessor1_Model03
4	0.000204	roc_auc	binary	0.859	1	NA	Preprocessor1_Model04
5	0.000259	roc_auc	binary	0.859	1	NA	Preprocessor1_Model05
6	0.000329	roc_auc	binary	0.859	1	NA	Preprocessor1_Model06
7	0.000418	roc_auc	binary	0.859	1	NA	Preprocessor1_Model07
8	0.000530	roc_auc	binary	0.859	1	NA	Preprocessor1_Model08
9	0.000853	roc_auc	binary	0.859	1	NA	Preprocessor1_Model10
10	0.00452	roc_auc	binary	0.860	1	NA	Preprocessor1_Model17
11	0.00574	roc_auc	binary	0.861	1	NA	Preprocessor1_Model18
12	0.00728	roc_auc	binary	0.861	1	NA	Preprocessor1_Model19
13	0.00924	roc_auc	binary	0.860	1	NA	Preprocessor1_Model20
14	0.0117	roc_auc	binary	0.860	1	NA	Preprocessor1_Model21
15	0.0149	roc_auc	binary	0.861	1	NA	Preprocessor1_Model22

Muestro la misma información anterior pero ordenada esta vez de forma decreciente por el valor ROC auc

Los datos se muestran ordenados de menor a mayor según el valor de penalty.

```
lr_res %>%
  show_best(metric = "roc_auc", n = 15) %>%
  arrange(desc(mean))
```

```
# A tibble: 15 x 7
  penalty .metric .estimator mean      n std_err .config
    <dbl> <chr>   <chr>    <dbl> <int>   <dbl> <chr>
1 0.00728 roc_auc binary    0.861     1     NA Preprocessor1_Model19
2 0.00574 roc_auc binary    0.861     1     NA Preprocessor1_Model18
3 0.0149  roc_auc binary    0.861     1     NA Preprocessor1_Model22
4 0.00924 roc_auc binary    0.860     1     NA Preprocessor1_Model20
5 0.0117  roc_auc binary    0.860     1     NA Preprocessor1_Model21
6 0.00452 roc_auc binary    0.860     1     NA Preprocessor1_Model17
7 0.0001  roc_auc binary    0.859     1     NA Preprocessor1_Model01
8 0.000127 roc_auc binary    0.859     1     NA Preprocessor1_Model02
9 0.000161 roc_auc binary    0.859     1     NA Preprocessor1_Model03
10 0.000204 roc_auc binary    0.859     1     NA Preprocessor1_Model04
11 0.000259 roc_auc binary    0.859     1     NA Preprocessor1_Model05
12 0.000329 roc_auc binary    0.859     1     NA Preprocessor1_Model06
13 0.000418 roc_auc binary    0.859     1     NA Preprocessor1_Model07
14 0.000530 roc_auc binary    0.859     1     NA Preprocessor1_Model08
15 0.000853 roc_auc binary    0.859     1     NA Preprocessor1_Model10
```

Observamos que el valor de penalty de 0.0072789538 aparece en el PUESTO 1.

Otra forma de observar esto es usando la función `select_best()` que encuentra la mejor combinación de hiperparámetros basándose en una medida de performance. En nuestro caso es un sólo hiperparámetro, penalty para el modelo de regresión logística, y la medida con la que evaluamos los modelos es ROC_AUC.

```
lr_res %>%
  select_best(metric = "roc_auc")
```

```
# A tibble: 1 x 2
  penalty .config
    <dbl> <chr>
1 0.00728 Preprocessor1_Model19
```

Nos indica que el modelo nro 19 es el mejorcito y el valor de penalty 0.007278954. Aunque si observamos con respecto a los otros valores de penalty menores, es muy poca la variación de ROC.

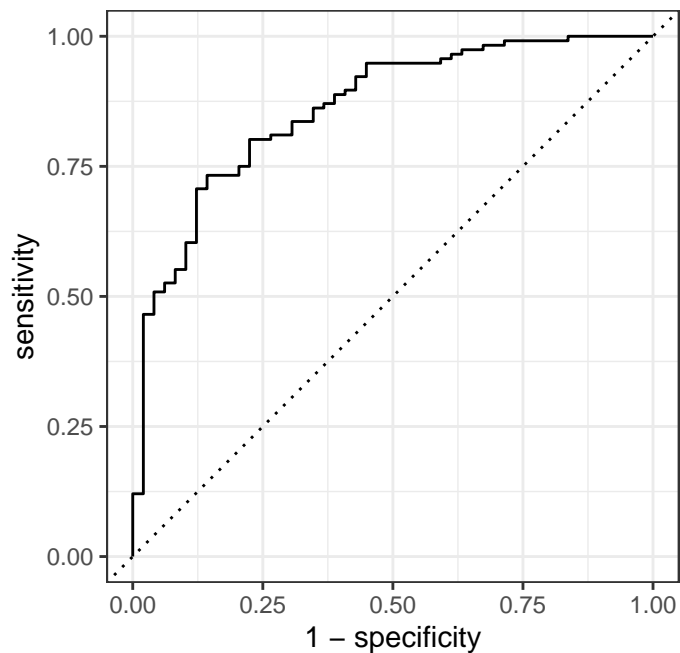
```
lr_best <- lr_res %>%  
  collect_metrics() %>%  
  arrange(desc(mean)) %>%  
  slice(1)  
lr_best
```

```
# A tibble: 1 x 7  
  penalty .metric .estimator mean      n std_err .config  
    <dbl> <chr>    <chr>    <dbl> <int>   <dbl> <chr>  
1 0.00728 roc_auc binary    0.861     1      NA Preprocessor1_Model19
```

Ahora tenemos nuestro modelo candidato: una logistic regresion con el valor de penalty mostrado arriba.

Graficamos la curva ROC para ese modelo con su performance en el conjunto de entrenamiento.

```
lr_auc <-  
  lr_res %>%  
  collect_predictions(parameters = lr_best) %>%  
  roc_curve(PM10, .pred_Bueno) %>%  
  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

RESULTADOS en conjunto de datos de testeo

Primero extraigo el mejor modelo, como ya explicamos antes.

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

best_model
```

```
# A tibble: 1 x 2
  penalty .config
  <dbl> <chr>
1 0.00728 Preprocessor1_Model19
```

Actualizamos nuestro workflow de trabajo.

```
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 = 0.00727895384398315
  mixture = 1

Computational engine: glmnet
```

Podemos utilizar la función `last_fit()` con nuestro modelo finalizado; esta función ajusta el modelo finalizado en el conjunto de datos de entrenamiento completo y evalúa el modelo finalizado en los datos de prueba.

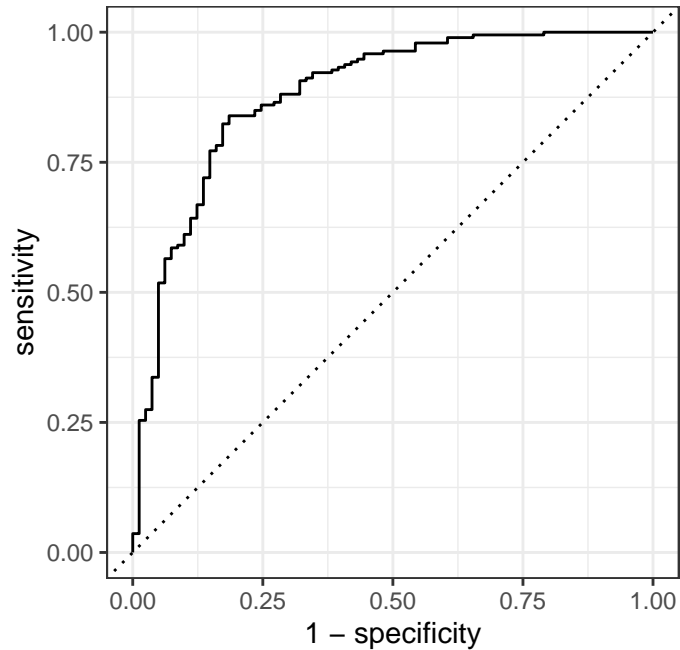
```
final_fit <-
  final_wf %>%
  last_fit(splits)
```

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

```
# A tibble: 3 x 4
  .metric      .estimator .estimate .config
  <chr>        <chr>         <dbl> <chr>
1 accuracy    binary         0.839 Preprocessor1_Model1
2 roc_auc     binary         0.881 Preprocessor1_Model1
3 brier_class binary         0.120 Preprocessor1_Model1
```


ROC CURVE SOBRE EL CONJUNTO DE TESTEO

```
final_fit %>%  
  collect_predictions() %>%  
  roc_curve(PM10, .pred_Bueno) %>%  
  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

Predecimos en el conjunto de datos de testeo, ese 25 % de datos que dejamos reservado para este momento.

```
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)
```

Matriz de confusión

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

	Truth	
Prediction	Bueno	Malo
Bueno	177	28
Malo	16	53

```
# check
nrow(data_test) == 177 + 16 + 28 + 53
```

```
[1] TRUE
```

Métricas de accuracy, sensitivity, specificity

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

```
# A tibble: 13 x 3
  .metric      .estimator .estimate
  <chr>        <chr>      <dbl>
1 accuracy    binary      0.839
2 kap         binary      0.597
3 sens        binary      0.917
4 spec        binary      0.654
5 ppv         binary      0.863
```

6 npv	binary	0.768
7 mcc	binary	0.601
8 j_index	binary	0.571
9 bal_accuracy	binary	0.786
10 detection_prevalence	binary	0.748
11 precision	binary	0.863
12 recall	binary	0.917
13 f_meas	binary	0.889

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      0.889
```

Random Forest

Vamos a entrenar un modelo de random forest para clasificación binaria.

Para ello usaremos la implementación de random forest de la librería ranger.

Durante el entrenamiento tunearemos dos hiperparámetros: mtry y min_n.

Dejamos trees en 100.

```
# nro de cores en el procesador de la COMPU donde esto se corre.
cores <- parallel::detectCores()

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

Receta: vamos a usar PM10 como variable a predecir que contiene los valores que queremos clasificar.

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

Armado del workflow tidymodels con el modelo y receta.

```
rf_workflow <-
  workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(rf_recipe)
```

Se tunea usando el conjunto de datos de entrenamiento, dentro del mismo establecemos una porción para validación, lo mismo que realizamos en el apartado anterior.

Se establece que la grilla sea de tamaño 25.

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

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

Podemos observar todos los resultados para cada valor de mtry y min_n ejecutando lo siguiente:

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

A tibble: 24 x 8

	mtry	min_n	.metric	.estimator	mean	n	std_err	.config
	<int>	<int>	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
1	4	7	roc_auc	binary	0.950	1	NA	Preprocessor1_Model01
2	8	30	roc_auc	binary	0.941	1	NA	Preprocessor1_Model02
3	3	25	roc_auc	binary	0.929	1	NA	Preprocessor1_Model03
4	7	34	roc_auc	binary	0.935	1	NA	Preprocessor1_Model04
5	4	4	roc_auc	binary	0.945	1	NA	Preprocessor1_Model05
6	5	10	roc_auc	binary	0.934	1	NA	Preprocessor1_Model06
7	1	38	roc_auc	binary	0.926	1	NA	Preprocessor1_Model07

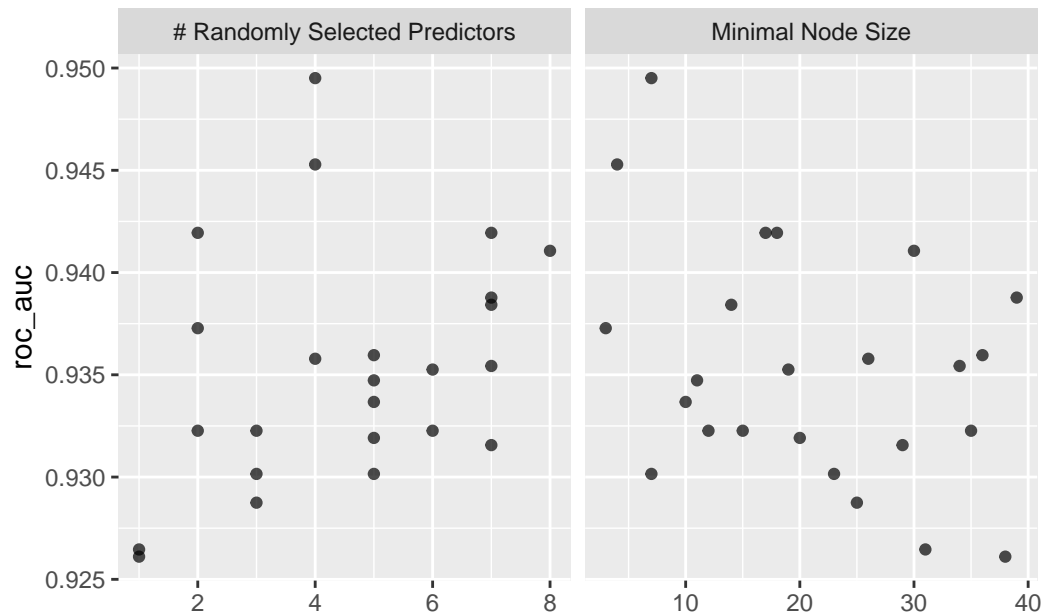
```

8      5      11 roc_auc binary    0.935    1      NA Preprocessor1_Model08
9      6      19 roc_auc binary    0.935    1      NA Preprocessor1_Model09
10     7      29 roc_auc binary    0.932    1      NA Preprocessor1_Model10
# i 14 more rows

```

Graficamos resultados del tuneo de hiperparámetros.

```
autoplot(rf_res)
```



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

Saving 1650 x 1050 px image

Observamos los mejores modelos.

```

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     4     7 roc_auc binary    0.950     1     NA Preprocessor1_Model01

```

2	4	4	roc_auc	binary	0.945	1	NA	Preprocessor1_Model05
3	7	17	roc_auc	binary	0.942	1	NA	Preprocessor1_Model21
4	2	18	roc_auc	binary	0.942	1	NA	Preprocessor1_Model23
5	8	30	roc_auc	binary	0.941	1	NA	Preprocessor1_Model02

Nos quedamos con el mejor modelo.

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

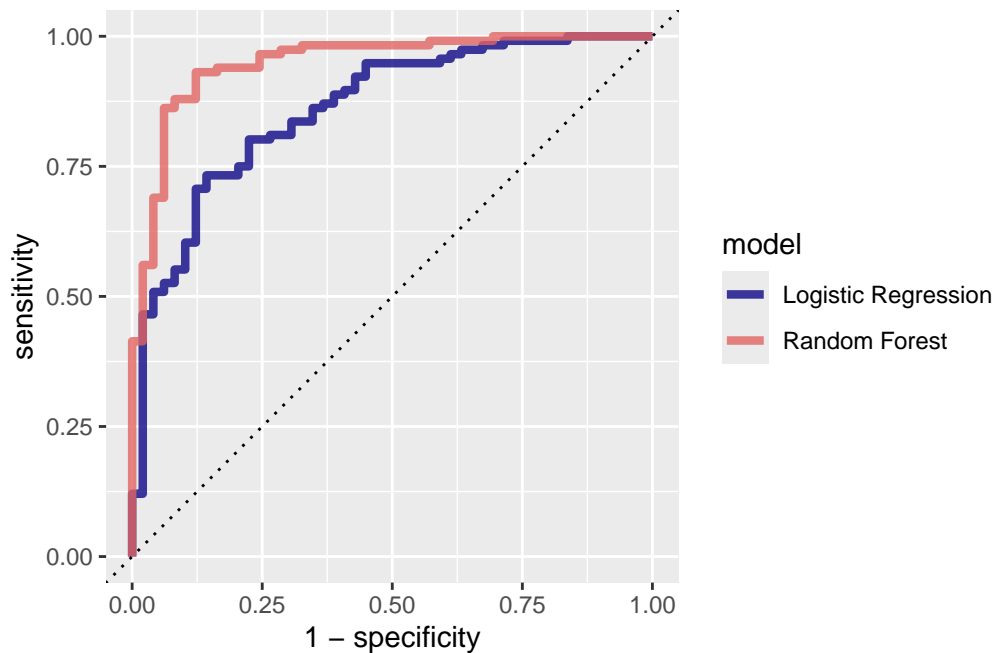
```
# A tibble: 1 x 3
  mtry min_n .config
  <int> <int> <chr>
1     4     7 Preprocessor1_Model01
```

Para filtrar las predicciones solo para nuestro mejor modelo, podemos usar el argumento de parámetros y pasarle nuestro tibble con los mejores valores de hiperparámetros del ajuste, al que llamamos rf_best:

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

ROC curve comparación de ambos modelos en el entrenamiento.

```
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

Observamos que el modelo de random forest es mejor en todo el umbral de probabilidad de eventos.

RESULTADOS FINALES

Armandos un modelo con los parámetros seleccionados, lo entrenamos, y luego queremos que prediga usando el conjunto de testeo.

```
# the last model
last_rf_mod <-
  rand_forest(mtry = 4, min_n = 7, 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)
```

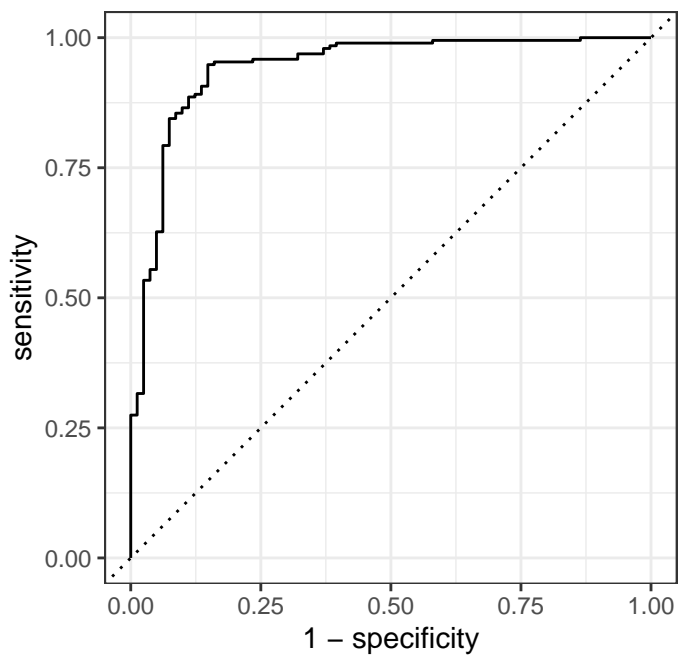
```
# the last fit
set.seed(345)
last_rf_fit <-
  last_rf_workflow %>%
  last_fit(splits)

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 [820/274]> train/test split <tibble> <tibble> <tibble>    <workflow>
```

ROC curve plot

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




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

Saving 1650 x 1050 px image

MATRIZ DE CONFUSION

```
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)
```

Matriz de confusión

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

	Truth	
Prediction	Bueno	Malo
Bueno	183	13
Malo	10	68

MÉTRICAS VARIAS

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

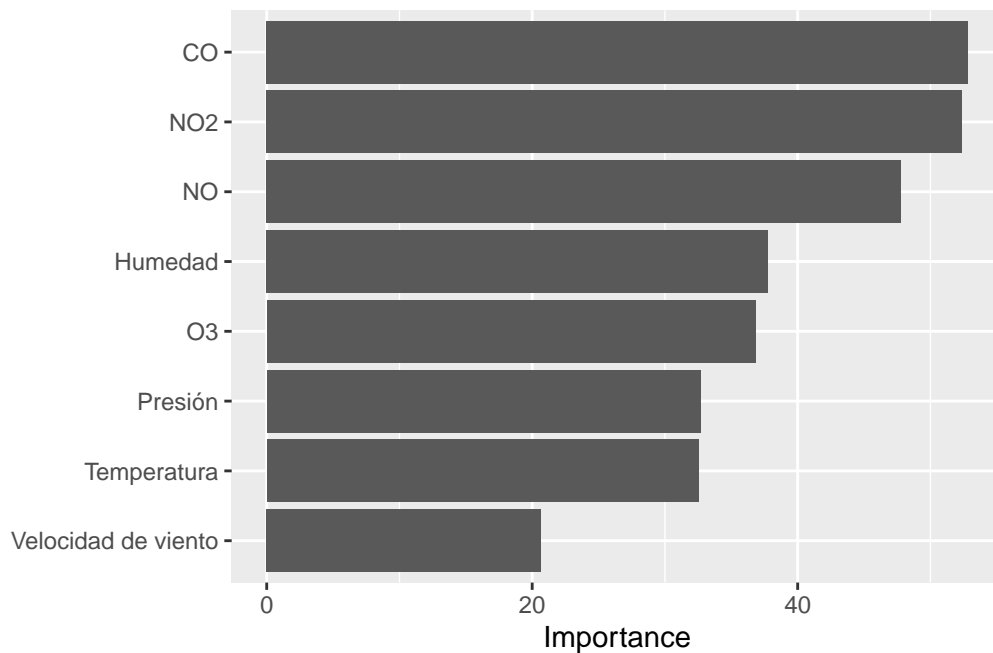
```
# A tibble: 13 x 3
```

	.metric <chr>	.estimator <chr>	.estimate <dbl>
1	accuracy	binary	0.916
2	kap	binary	0.796
3	sens	binary	0.948
4	spec	binary	0.840
5	ppv	binary	0.934
6	npv	binary	0.872
7	mcc	binary	0.797
8	j_index	binary	0.788
9	bal_accuracy	binary	0.894
10	detection_prevalence	binary	0.715
11	precision	binary	0.934
12	recall	binary	0.948
13	f_meas	binary	0.941

Vip Variable importance

Listamos de mayor a menor importancia las variables. Significa las variables que más colaboran en la predicción del modelo RF.

```
last_rf_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 10)
```



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

Saving 1650 x 1050 px image

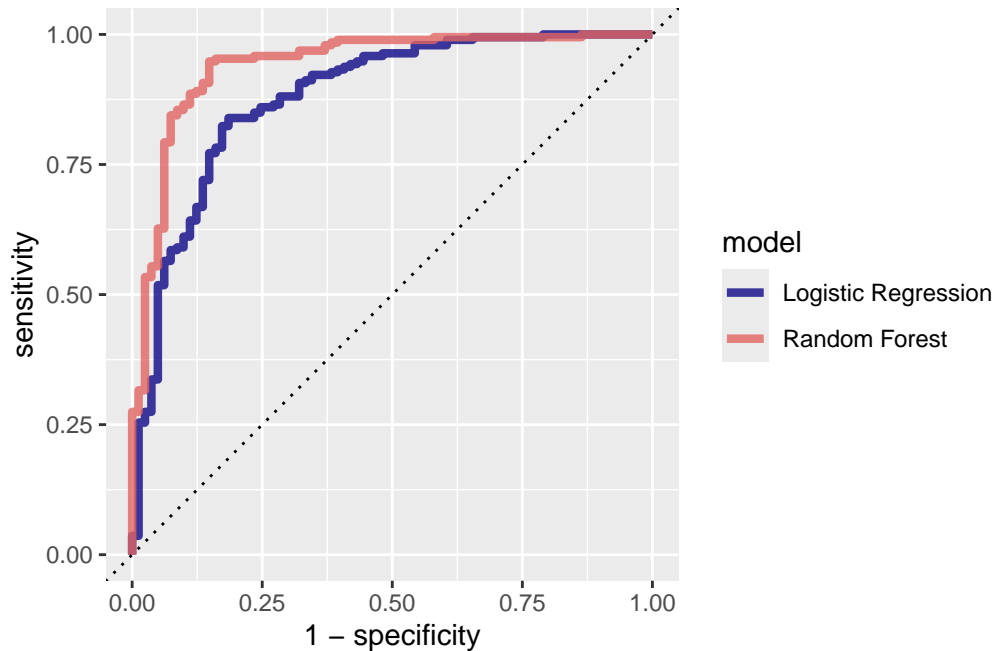
COMPARACION

ROC CURVE

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

```
rf_auc_f <- last_rf_fit %>%
  collect_predictions() %>%
  roc_curve(PM10, .pred_Bueno) %>%
  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

METRICAS

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

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

TABLA COMPARATIVA METRICAS POR CADA MODELO

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

A tibble: 13 x 3

.metric	`Random Forest`	`Logistic Regression`
<chr>	<dbl>	<dbl>
1 accuracy	0.839	0.916
2 kap	0.597	0.796
3 sens	0.917	0.948
4 spec	0.654	0.840
5 ppv	0.863	0.934
6 npv	0.768	0.872
7 mcc	0.601	0.797
8 j_index	0.571	0.788
9 bal_accuracy	0.786	0.894
10 detection_prevalence	0.748	0.715
11 precision	0.863	0.934
12 recall	0.917	0.948
13 f_meas	0.889	0.941

Guardo los resultados en un archivo csv

```
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")
```