Índice general

Capítulo 1

Apéndice: Código

1.1. Generación de la muestra

```
# Librerías ------------------------------------------------------------
# Se cargan las librerías que se usarán en esta sección

library(terra) # Raster data
library(sf) # Vector data
library(mapSpain) # Polígonos de las regiones de España
library(tidyverse) # Manipulación de datos



# CRS de referencia ----------------------------------------------------
# Será el CRS que se use en todo el proyecto

pend <- rast("data_raw/topograficas/pendiente.tif")
crs_reference = crs(pend)
rm(pend) # Se elimina de la memoria para liberar espacio


# Polígono de Andalucía ------------------------------------------------
Andalucia <- esp_get_ccaa(ccaa = "Andalucía") # Se obtiene el polígono de la comunid
andalucia_proj <- st_transform(Andalucia,crs_reference) # Se transforma al sistema de

# area_monte es el área donde se generarán las muestras negativas.

# Dado que no hay un mapa que indique claramente cuales son las zonas que se conside
area_monte <- andalucia_proj

# Generación de la muestra ---------------------------------------------

# Generación de la muestra estratificando por mes de forma que la proporción de obse
```

```r
##  Tamaño muestral ----------------------------------------------------
# Se dispone de 1089 incencios correctamente registrados entre 2002 y 2022

n_in=10 # Número de puntos a muestrear dentro de cada poligono
n_out=1089*10 # Número de muestras negativas

##  Generación aleatoria de fechas para las muestras negativas ----------

# Primero se leen todos los datos de todos los archivos de incendios y se almacenan
incendios = NULL

for (year in 2002:2022) {
  incendios = rbind(incendios,
                    st_read(paste0("./data_raw/incendios_2000-2022/incendios_",year,"
                      select("FECHA_INIC" = matches("(?i)^FECHA_INIC$|^fecha_inic.$")
}

# Se cuenta el número de incendios con fecha de inicio correcta en cada mes
incendios_mes = incendios %>%
  mutate(FECHA_INIC = ymd(FECHA_INIC),.keep="unused") %>%
  filter(!is.na(FECHA_INIC)) %>%
  filter(year(FECHA_INIC)<=2022,year(FECHA_INIC)>=2002) %>%
  st_drop_geometry() %>%
  mutate(MES = month(month(FECHA_INIC))) %>%
  count(MES)

# Fechas posibles para las muestras negativas
possible_dates = tibble (date = seq(as.Date('2002/01/01'), as.Date('2022/12/31'), by=
  mutate(MES = month(date)) %>%
  left_join(incendios_mes,
            join_by(MES))

set.seed(12345) # Se fija la semilla para que sea reproducible

# Se generan fechas aleatorias para las muestras negativas entre 2002 y 2022 siguien
dates = sample(possible_dates$date,
               n_out,replace = T,
               prob = possible_dates$n)

rm(incendios, possible_dates) # Se borran para liberar memoria

##  Selección de localizaciones aleatorias -----------------------------
# Para la selección de la muestra se seguirá el siguiente procedimiento:
# 1. Para las muestra positivas: Se tomarán n_in puntos aleatorios dentro de cada po
# 2. Para la muestras negativas: Se le asociará una localización aleatoria dentro de

points_in = NULL  # Almacena las muestras positvas
points_out = NULL # Almacena las muestras negativas
```

```
for (year in 2002:2022) {

  cat("YEAR ", year," : ------------------------------------\n")
  cat("  Generando muestras positivas...\n")
  incendios <- st_read(paste0("./data_raw/incendios_2000-2022/incendios_",year,".shp"
    st_transform(crs = crs_reference) |>
    rename_with(.fn=tolower) |>
    mutate(fecha_inic=ymd(fecha_inic),geometry,.keep="none")


  ## Generación de puntos positivos

  for (i in 1:nrow(incendios)) {
    point_in_sfc <- st_sample(incendios[i,],size=n_in) # Se generan n_i puntos dentro
    point_in_attr <- data.frame(fire = rep(1,n_in),date = rep(incendios[i,]$fecha_ini
    point_in <- st_sf(point_in_attr,geometry= point_in_sfc)

    if (is.null(points_in)) {
      points_in <- point_in
    } else {
      points_in <- points_in |>
        add_row(point_in)
    }
  }

  ## Generación de puntos negativos

  cat("  Generando muestras negativas...\n")
  # ---> Nota: los puntos se generan en area_monte

  dates_year <- dates[year(dates) == year]
  locations = NULL

  for (day in dates_year) {
    incendios_day = filter(incendios,fecha_inic>=day-3 & fecha_inic<=day+3)
    if (nrow(incendios_day)==0){ # Si no ha habido incendios en una franja de 6 días
      if (is.null(locations)) {
        locations = st_sample(area_monte,size=1)
      } else {
        locations = c(locations, st_sample(area_monte,size=1))
      }
    } else { # Si ha habido algún incendio en una franja de 6 días en Andalucía (3 d
      repeat {
        possible_location = st_sample(area_monte,size=1)
        if (!st_is_within_distance(possible_location,st_union(incendios_day), dist =
          possible_location = st_sample(area_monte,size=1)
          if (is.null(locations)) {
```

```
            locations = possible_location
            break
          } else {
            locations = c(locations, possible_location)
            break
          }
        }
      }
    }
  }


  points_out_attr <- data.frame(fire = rep(0,length(dates_year)),date = dates_year)

  if (is.null(points_out)) {
    points_out <- st_sf(points_out_attr,geometry= locations)
  } else {
    points_out <- points_out |>
      add_row(st_sf(points_out_attr,geometry= locations))
  }

}


sample <- rbind(points_in,points_out) # La muestra generada


# Comprobación y corrección ----------------------------------------------
summary(sample) # Hay una fecha de un incendio errónea
max(sample$date,na.rm=T) # "2033-08-15"

# Se eliminan las observaciones con fecha de incendio errónea que se han detectado
sample <- sample[-which(sample$date==max(sample$date,na.rm=T)),]
summary(sample) # Corregido

# Almacenamiento de resultados -------------------------------------------
save(sample,file=paste0("salidas_intermedias/sample_strat_",Sys.Date(),".RData"))
```

1.2. Asignación de variables a localizaciones

A continuación se define la función `asignar_variables` que dada una muestra de puntos en Andalucía con fechas comprendidas entre 2002 y 2022 le asocia a cada observación todos los valores de las variables consideradas en el estudio. Esta función se usará varias veces a lo largo del trabajo.

```r
# Librerías -----------------------------------------------------------
# Se cargan las librerías que se usarán en esta sección
library(nasapower) # Para obtener la información meteorológica
library(raster, include.only = c("rasterFromXYZ"))  # Función para construir rasters
library(tidyverse)
library(sf)
library(terra)
library(mapSpain)

asignar_variables = function(sample) {
  # Argumentos:
  # * sample: objeto sf con una columna de geometrías de tipo POINT (dentro de los l


  crs_reference = st_crs(sample) # Se usa el sistema de referencia de coordenadas de
  and = esp_get_ccaa(ccaa = "Andalucía") %>% st_transform(st_crs(datos)) # Polígono

  # Variables meteorológicas ------------------------------------------------
  cat("Asignando variables meteorológicas...\n")

  # Tranformamos los datos a WGS84
  andalucia_WGS84 <- st_transform(and,crs="WGS84")

  dataset = NULL # Variable en la que se almacenará el conjunto completo

  # Se trabaja anualmente pues la API de NASA POWER solo admite consultas de hasta 3

  for (year in sort(unique(year(sample$date)))) {

    cat("YEAR ", year," : ------------------------------------\n")

    # Los puntos de cada año
    points = filter(sample,year(date)==year)
    points_WGS84 <- st_transform(points,crs="WGS84")

    # Consulta a la api para obtener todo los valores del año
    daily_single_ag <- get_power(
      community = "ag",
      lonlat = c(-8,35.5,-1.5,39),  # Límites de Andalucía
      pars = c("T2M","GWETTOP", "RH2M","WD10M","WS10M","PRECTOTCORR"),
      dates = paste0(year,c("-01-01","-12-31")),
      temporal_api = "daily")

    # Identificador
    daily_single_ag$clim_id <- 1:nrow(daily_single_ag)
    points$clim_id = NA # Inicializo el identificador

    for (day in unique(points$date)) {
```

```
    points_day = points$date==day

    # Seleccionar un día
    clim_day  <- filter(daily_single_ag,YYYYMMDD==day) |>
      dplyr::select(x = LON,y = LAT,clim_id= clim_id)

    id_rast_day = rast(rasterFromXYZ(clim_day,crs="WGS84")) # Se crea el raster co

    points[points_day,]$clim_id <- terra::extract(id_rast_day,points_WGS84[points_d
  }

  # Haciendo uso del identificador se asocian todas las variables meteorológicas
  points <- points |>
    left_join(select(daily_single_ag, -c(LAT,LON,DOY,YYYYMMDD)),
              by=join_by(clim_id)) |>
    select(-clim_id)

  dataset = rbind(dataset,points)
}

rm(points,points_WGS84,daily_single_ag,clim_day,id_rast_day,points_day,day,year,anda


# Variables topográficas ----------------------------------------------------
cat("Asignando variables topográficas...\n")
elev <- rast("data_raw/topograficas/elevacion.tif")
pend <- rast("data_raw/topograficas/pendiente.tif")
orient <- rast("data_raw/topograficas/orientacion.tif")
curv <- rast("data_raw/topograficas/curvatura.tif")

# Es necesario pasarlas a numeric para poder trabajar con ellas y extraer los valo
pend <- as.numeric(pend)
orient <- as.numeric(orient)
curv <- as.numeric(curv)

# Se extraen los valores de cada una de las capas
var_topograficas <- list(elevacion = elev,pendiente = pend,orientacion = orient,cur

points_topograficas <- sapply(var_topograficas,function(x) terra::extract(x,dataset
  as_tibble()

dataset <- cbind(dataset,points_topograficas)

rm(elev,pend,orient,curv,var_topograficas,points_topograficas)


# Variables antropológicas --------------------------------------------------
```

```r
cat("Asignando variables antropológicas...\n")

## Para optimizar el cálculo evitando que se repitan cálculos si hay puntos repeti
dataset_geoms <- dataset %>%
  group_by(geometry) %>%
  group_keys() %>%
  st_sf(crs = st_crs(dataset))

### Carreteras: ----
carreteras <- read_sf("data_raw/antropologicas/RedCarreteras/09_14_RedCarreteras.sh
  st_union()

dataset_geoms$dist_carretera <- st_distance(dataset_geoms,carreteras) |>
  as.numeric()      # metres

rm(carreteras)

### Poblaciones: ----
poblaciones <- read_sf("data_raw/antropologicas/Poblaciones/07_01_Poblaciones.shp")
  st_union()

dataset_geoms$dist_poblacion <- st_distance(dataset_geoms,poblaciones) |>
  as.numeric()    # metres

rm(poblaciones)

### Linea Eléctrica: ----
linea_electrica <- read_sf("data_raw/antropologicas/LineaElectrica/10_14_LineaElect
  st_union()

dataset_geoms$dist_electr <- st_distance(dataset_geoms,linea_electrica) |>
  as.numeric() # metres

rm(linea_electrica)

### Ferrocarril: ----
ferrocarril <- read_sf("data_raw/antropologicas/Ferrocarril/09_21_Ferrocarril.shp")
  st_union()
dataset_geoms$dist_ferrocarril <- st_distance(dataset_geoms,ferrocarril) |>
  as.numeric()

rm(ferrocarril)

### Camino / Via: ----
camino <- read_sf("data_raw/antropologicas/Camino/09_19_Camino.shp")
viapec <- read_sf("data_raw/antropologicas/Camino/09_22_ViasPecuarias.shp")

camino_viapec <- c(st_geometry(camino),st_geometry(viapec))
```

```r
rm(camino,viapec)

camino_viapec <- st_union(camino_viapec)

dataset_geoms$dist_camino <- st_distance(dataset_geoms,camino_viapec) |>
  as.numeric()

rm(camino_viapec)

### Sendero / Vía Verde / CarrilBici: ----
viaverde <- read_sf("data_raw/antropologicas/Sendero_ViaVerde/09_24_ViaVerde.shp")
sendero <- read_sf("data_raw/antropologicas/sendero_ViaVerde/09_20_Sendero.shp")
carrilbic <- read_sf("data_raw/antropologicas/sendero_ViaVerde/09_23_CarrilBici.shp

sendero_viaverde_carrilbici <- c(st_geometry(viaverde),st_geometry(sendero),st_geom
  st_union()

dataset_geoms$dist_sendero <- st_distance(dataset_geoms,sendero_viaverde_carrilbici
  as.numeric()

rm(sendero,sendero_viaverde_carrilbici,viaverde,carrilbic)

### ENP: ----
enp1 <- read_sf("data_raw/antropologicas/ENP/11_07_Enp_FiguraProteccion.shp" )
enp2 <- read_sf("data_raw/antropologicas/ENP/11_07_Enp_RegimenProteccion.shp")

enp <- c(st_geometry(enp1),st_geometry(enp2)) |>  st_union()
enp_sf <- st_sf(enp)

# Se rasteriza para aumentar la eficiencia computacional
enp_rast <- rasterize(enp_sf,
                      rast("data_raw/topograficas/pendiente.tif"), # Modelo
                      background = 0)
dataset_geoms$enp= terra::extract(enp_rast,dataset_geoms)[,2]

rm(enp,enp1,enp2,enp_sf,enp_rast)

### Uso Suelo: ----
# Inicialmente se ha rasterizado para aumentar la eficiencia computacional
# UsoSuelo <- read_sf("data_raw/antropologicas/UsoSuelo/06_01_UsoSuelo.shp")
# UsoSuelo_rast <- rasterize(UsoSuelo,
#                            rast("data_raw/topograficas/pendiente.tif"), # Modelo
#                            field="cod_uso")

UsoSuelo_rast <- rast("data_cleaning/uso_suelo_rast.tiff")

dataset_geoms$uso_suelo = terra::extract(UsoSuelo_rast,dataset_geoms)[,2]
```

```r
# Hidrográficas ------------------------------------------------------------
cat("Asignando variables hidrográficas...\n")

### Distancia a ríos: ----
rios <- read_sf("data_raw/hidrograficas/Rios_Espana.shp") |>
  st_transform(st_crs(dataset)) |>
  st_crop(xmin = 100394.4, # Esto se hace solo para no tener que considerar todo e
          ymin = 3976888.6,
          xmax = 690000.8,
          ymax = 4350000.0) |>
  st_union()

dataset_geoms$dist_rios <- st_distance(dataset_geoms,rios) |>
  as.numeric() # metres

rm(rios)

##  Se vuelven a desagrupar los registros y se le asigna a cada registro los valor
dataset <- dataset %>%
  st_join(dataset_geoms,left = TRUE) # Es un left join espacial

# Demográficas ------------------------------------------------------------
cat("Asignando variables demográficas...\n")

### Población y densidad de población: ----

poblacion <- read_csv2("data_raw/antropologicas/Población/poblacion_municipios.txt"
                       locale=locale(decimal_mark = ","),
                       col_select = 1:5,col_types = "ccifn") |>
  mutate(Valor=as.integer(round(Valor))) # Por algún motivo aparecen decimales


area_municipios <- read_csv2("data_raw/antropologicas/Población/extension_municipal
                             locale=locale(decimal_mark = ","),
                             col_select = 1:6, col_types = "fffffn")

area_municipios <- area_municipios %>%
  filter(!is.na(CODIGO_INE3)) %>%
  select(CODIGO_INE3,Valor) %>%
  rename("Area" = "Valor")

# Se calcula la densidad de población anual como el cociente del número de habitar
dens_poblacion <- poblacion %>%
  select(-Medida) %>%
  rename("Poblacion" = "Valor",
         "Municipio" = "Lugar de residencia") %>%
  left_join(area_municipios,
            join_by("CODIGO_INE3")) %>%
```

```
  mutate(dens_poblacion = Poblacion/Area) %>%
  select(-Area)

municipios <- esp_get_munic(epsg = 4258,region = "Andalucía")

municipios <- municipios |>
  st_transform(crs_reference)

# Se asocia cada observacion su código de municipio correspondiente

num_mun = st_intersects(dataset,municipios)

# Se eliminan las observaciones que no están en ningún municipio
if (any(sapply(num_mun,function(x) length(x) == 0))) {
  cat("Eliminamos las observaciones:\n",which(sapply(num_mun,function(x) length(x)
  dataset = dataset[-which(sapply(num_mun,function(x) length(x) == 0)),]
}

dataset$cod_municipio <- municipios[unlist(st_intersects(dataset,municipios)),]$LAU

dataset <- dataset |>
  left_join(dens_poblacion,
            join_by(cod_municipio==CODIGO_INE3,YEAR==Anual)) |>
  rename("municipio" = "Municipio",
         "poblacion" = "Poblacion")

# Vegetación ------------------------------------------------------------------
cat("Asignando variables de vegetación...\n")

### NDVI ----
dataset$NDVI = NA

for (YEAR in 2002:2022) {
  for (MONTH in 1:12) {
    MM = str_pad(MONTH,2,"left",pad = "0")
    YY = substr(as.character(YEAR),3,4)
    # if (as.numeric(YY)<=01) {
    #   ruta <- paste0("data_raw/vegetacion/",YEAR,"NOAAVHMEDMNDVI/InfGeografica/I
    # } else
    if (as.numeric(YY)<=06) {
      ruta <- paste0("data_raw/vegetacion/",YEAR,"TERMODMEDMNDVI/InfGeografica/InfR
    } else if (as.numeric(YY)<=11) {
      ruta <- paste0("data_raw/vegetacion/",YEAR,"TERMODMEDMNDVI/InfGeografica/InfR
    } else if (as.numeric(YY)<=21) {
      ruta <- paste0("data_raw/vegetacion/",YEAR,"TERMODMEDMNDVI/InfGeografica/InfR
    }else {
      ruta <- paste0("data_raw/vegetacion/",YEAR,"TERMODMEDMNDVI/InfGeografica/InfR
    }
```

```
      if (file.exists(ruta)) {
        cat(YEAR,MONTH,"\n")
        # Observaciones en ese mes y año
        isMY = dataset$YEAR==YEAR & dataset$MM==MONTH
        if (any(isMY)) {
          NDVI_rast = as.numeric(rast(ruta))
          if (MONTH==4 & YEAR==2011){
            # Ese archivo viene defectuoso y se le asigna el CRS de los otros archiv
            crs(NDVI_rast) = "PROJCRS[\"WGS 84 / UTM zone 30N\",\n    BASEGEOGCRS[\"W
          }

          dataset[isMY,]$NDVI = terra::extract(NDVI_rast,dataset[isMY,])[,2]
        }
      } else
        cat("No existe: ",YEAR,"-",MONTH,"\n")
    }
  }


  # Factores ------------------------------------------------------------
  # Codificación de las variables categóricas como factores:

  dataset <- dataset |>
    mutate(enp = as.factor(enp),
           orientacion = cut(orientacion,
                             breaks = c(-Inf,-1,22.5,67.5,112.5,157.5,202.5,247.5,292
                             labels = c("Plano","N","NE","E","SE","S","SW","W","NW","
           WD10M = cut(WD10M,
                       breaks = c(0,22.5,67.5,112.5,157.5,202.5,247.5,292.5,337.5,360
                       labels = c("N","NE","E","SE","S","SW","W","NW","N")),
           uso_suelo = uso_suelo |> as.character() |> str_sub(0,2) |> as.factor()
           ) %>%
    select(-c(YEAR,MM,DD))

  return(dataset)
}
```

Se usa la función definida para asignar las variables explicativas a la muestra generada:

```
# Se carga la muestra generada en el paso anterior:
load("salidas_intermedias/sample_strat_2024-04-26.RData")

# Se eliminan las observaciones que no tienen fecha pues se pueden usar para el estu
sample <- na.omit(sample)

# Se aplica la función a la muestra
dataset <- asignar_variables(sample)
```

```
# Se almacenan los resultados
save(dataset,file = paste0("salidas_intermedias/dataset_strat_completo",Sys.Date(),".
```

1.3. Depuración de la muestra

En la propia función usada para generar la muestra ya se ajustaron los tipos de las variables y se codificaron adecuadamente las variables *WD10M* y *orientacion*. A continuación se estudian los casos faltantes y finalmente se decide eliminarlos.

```
# Librerías ---------------------------------------------------------------
# Se cargan las librerías que se usarán en esta sección
library(tidyverse) # Manipulación de datos
library(sf) # Vector data
library(terra) # Raster data
library(mapSpain) # Polígonos de regiones de España
library(magrittr) # Operador %<>%
library(skimr) # Resumen de datos

# Carga de los datos ------------------------------------------------------
# Se carga la muestra con todas las variables construida anteriormente
load("salidas_intermedias/dataset_strat_completo2024-04-26.RData")


# Codificación variable fire: ---------------------------------------------
datos <- dataset |>
  mutate(fire = as.factor(fire))

# Análisis de valores perdidos: -------------------------------------------
datos %>% skimr()
dataset %>% apply(1,anyNA) %>% sum() # 200 registros con valores perdidos

# Se observan:
#   8 nas en uso_suelo
#   32 nas en pendiente
#   36 nas en orientacion
#   32 nas en curvatura
#   85 nas en doblacion y en dist_poblacion
#   85 nas en NDVI

datos1 = datos %>% drop_na()

# Valores perdidos en variables topográficas ------------------------------
Andalucia <- esp_get_ccaa(ccaa = "Andalucía") # Se carga el mapa de Andalucía
andalucia_proj <- st_transform(Andalucia,st_crs(dataset)) # Se proyecta al sistema d

# Mapa de valores perdidos en variables topográficas:
```

```r
plot(st_geometry(andalucia_proj),reset=F)
datos |>
  filter(is.na(pendiente)| is.na(orientacion) | is.na(curvatura)) |>
  st_geometry() |>
  plot(pch=16,col="red",add=T) # Se encuentran en los límites de Andalucía

datos |>
  st_drop_geometry() |>
  filter(is.na(pendiente)| is.na(orientacion) | is.na(curvatura)) |>
  count() #53 registros con algún valor perdido entre las variables topográficas


# Valores perdidos en variables demográficas ----------------------------
# Se observa que los valores perdidos se deben a que los datos no estan disponibles
datos |>
  st_drop_geometry() |>
  filter(is.na(poblacion)) |>
  mutate(Año = year(date)) |>
  select(Año,municipio,cod_municipio)

# Ejemplo:
pob = read_csv2("data_raw/antropologicas/Población/poblacion_municipios.txt")[,c(1:5)
pob |> filter(CODIGO_INE3 == "18077") # Datos no disponibles

datos |>
  st_drop_geometry() |>
  filter(is.na(poblacion)) |>
  count() # 85 resgistros con valores perdidos en las variables demográficas

# Valores perdidos en uso_suelo -----------------------------------------
dataset |> filter(is.na(uso_suelo)) %>% count() # 8 registros con valores perdidos

dataset |> filter(is.na(uso_suelo)) |> st_geometry() |> plot(pch=16,col="red",add=T)
# De nuevo se observa que se encuentran en los límites de Andalucía con la costa

# Valores perdidos en NDVI ----------------------------------------------
# Se muestran en el mapa
plot(st_geometry(andalucia_proj),reset=F)
dataset |> filter(is.na(NDVI)) |> st_geometry() |> plot(pch=16,col="red",add=T)


# Gráfico de NDVI medio cada mes
NDVI_mes = datos |>
  st_drop_geometry() |>
  mutate(mes = month(date),
         año = year(date)) |>
  group_by(año,mes) |>
  summarise(NDVI_mes = mean(na.omit(NDVI))) |>
```

```
  ungroup() |>
  mutate(date = dmy(paste(01,mes,año,sep="/")))

NDVI_mes |>
  ggplot(aes(x=date,y=NDVI_mes)) +
  geom_line()+
  scale_x_date(date_breaks = "6 month",date_labels = "%y%b")+
  theme(axis.text.x = element_text(angle = 90))

# 2008 tiene un NDVI especialmente bajo, puede deberse a numerosos factores:
#   - Que efectivamente sea un año especialmente seco
#   - Las características de la muestra seleccionada ese año
#   - Errores de medición
#   - ...

# La mayor parte de los valores perdidos en la variable uso_suelo son debidos a que

# Eliminación registros con valores perdidos ------------------------------
# Tras explorar otras opciones finalmente se opta por eliminar los registros
# que contienen valroes perdidos:
datos = datos %>% drop_na()

# Almacenamiento del conjunto de datos depurados --------------------------
# Se guarda el conjunto de datos depurados
save(datos, file = paste0("salidas_intermedias/datos_strat_depurados_geom_",Sys.Date(
```

1.4. Análisis exploratorio

```
# Librerías ---------------------------------------------------------------
# Se cargan las librerías que se usarán en esta sección
library(tidyverse) # Manipulación de datos
library(sf) # Vector data
library(terra) # Raster data
library(mapSpain) # Polígonos de regiones de España
library(magrittr) # Operador %<>%
library(skimr) # Resumen de datos
library(corrplot) # Gráfico de coorrelaciones
library(GGally) # Coordenadas paralelas
library(ggpubr) # Función ggarrange

# Carga de los datos ------------------------------------------------------
load("salidas_intermedias/datos_strat_depurados_geom_2024-05-03.RData")

# Resumen numérico --------------------------------------------------------
datos %>%  st_drop_geometry %>% skim(.data_name = "datos")
```

1.4.1. Variable objetivo:

```
# Distribución temporal ------------------------------------------------
# Mensual:
g_mes <- datos %>%
  st_drop_geometry() %>%
  # filter(fire==1)%>%
  count(month(date,label=T),fire) %>%
  rename("mes" = "month(date, label = T)") %>%
  ggplot(aes(x = mes,y = n,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  # theme(axis.text.x = element_text(angle = 90)) +
  scale_fill_hue(direction = -1) +
  theme_minimal() +
  xlab("Mes") +
  ylab("Número de observaciones")

g_mes

# Anual
g_year <- datos %>%
  st_drop_geometry() %>%
  # filter(fire==1)%>%
  count(year(date),fire) %>%
  rename("año" = "year(date)") %>%
  ggplot(aes(x = año,y = n,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_x_continuous(breaks=2002:2022) +
  geom_smooth(se=F,aes(color=fire),alpha=0.1) +
  scale_color_manual(values=c("darkblue","darkred")) +
  scale_fill_hue(direction = -1) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  xlab("Año") +
  ylab("Número de observaciones")
g_year

# En 2007 no se han generado observaciones positivas
# Causa: En el shapefile que contiene los incendios producidos en 2007 solo hay 4 ob

# Valores muy bajos en 2008 y 2010, revisar
# 2010: correcto, en el archivo con los polígonos de incendios en 2010 solo hay 26 d
# 2008: en el archivo relativo a ese año hay 37 observaciones, pero 11 de ellas no t

# Día de la semana
g_dia <- datos %>%
  st_drop_geometry() %>%
```

```
  # filter(fire==1) %>%
  count(weekdays(date),fire) %>%
  rename("dia" = "weekdays(date)") %>%
  ggplot(aes(x = dia,y = n,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_x_discrete(limits=c("lunes", "martes", "miércoles", "jueves", "viernes", "sáb
  theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=0.5)) +
 # geom_smooth(se=F) +
  scale_fill_hue(direction = -1) +
  theme_minimal() +
  xlab("Día") +
  ylab("Número de observaciones")
g_dia

# Para mostrar los tres gráficos juntos
ggarrange(g_dia,g_mes,g_year,ncol=1,common.legend = T,legend = "bottom")

# Distribución temporal ----------------------------------------------------
# Se cargan los polígonos de las provincias
and <- esp_get_ccaa(ccaa = "Andalucía") %>% st_transform(st_crs(datos))
prov <- esp_get_prov() %>% filter(nuts2.name=="Andalucía") %>% st_transform(st_crs(

# Casos positivos
g1 = ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos %>% filter(fire==1),size = 1, alpha = 0.4,col="red")+
  ggtitle("Distribución espacial de casos positivos") +
  theme_minimal()

# Casos negativos
g0 = ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos %>% filter(fire==0),size = 1, alpha = 0.4,col="blue") +
  ggtitle("Distribución espacial de casos negativos") +
  theme_minimal()

# Ambos gráficos juntos:
ggarrange(g1,g0,nrow=2)
```

1.4.2. Análisis univariantes

1.4.2.1. T2M

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=T2M),size = 1.2, alpha = 0.6) +
```

```
  facet_wrap(~month(date,label=TRUE)) +
  scale_color_gradientn(colours = rainbow(5,rev=T)) +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de T2M por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Las temperaturas más elevadas se encuentran en el interior.

```
datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
  summarize(AVG_T2M = mean(T2M),
            Mes = `month(date, label = T)`) %>%
  ggplot(aes(x=Mes,y=AVG_T2M,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_fill_hue(direction = -1) +
  theme_minimal()
  # + theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=0.5))
```

En prácticamente todos los meses, la media de temperaturas en las observaciones positivas es mayor que en las observaciones negativas. Las temperaturas son más altas en los meses de verano, como cabía esperar.

1.4.2.2. RH2M

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=RH2M),size = 1.2, alpha = 0.6) +
  facet_wrap(~month(date,label=TRUE)) +
  scale_color_gradientn(colours = rainbow(5,rev=F)) +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de RH2M por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

La humedad del aire es mayor en las de costa. Las zonas más secas se encuentran en las zonas de interior del norte de Andalucía.

```
datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
  summarize(AVG_RH2M = mean(RH2M),
            Mes = `month(date, label = T)`) %>%
  ggplot(aes(x=Mes,y=AVG_RH2M,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_fill_hue(direction = -1) +
  theme_minimal()
  # + theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=0.5))
```

En prácticamente todos los meses, la humedad del aire media entre las observaciones positivas es menor que en las observaciones negativas. La humedad del aire disminuye significativamente en verano.

1.4.2.3. GWETTOP

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=GWETTOP),size = 1, alpha = 0.6) +
  facet_wrap(~month(date,label=TRUE)) +
  scale_color_gradientn(colours = rainbow(5,rev=F)) +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de GWETTOP por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Las zonas con mayor humedad superficial de suelo son las que se encuentran en la costa mediterránea. Las zonas con un suelo más seco son las que se encuentran al norte de la cuenca el Guadalquivir, destacando la provincia de Huelva (hay que tener en cuenta que también es una zona en la que hay muchos incendios, la mayoría de los cuales se produce en verano, por lo que hay una gran concentración de observaciones durante el periodo estival, lo que podría influir en el hecho de que se observen tantas observaciones con valores tan bajos de la humedad superficial de suelo).

```
datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
```

```
  summarize(AVG_GWETTOP = mean(GWETTOP),
            Mes = `month(date, label = T)`) %>%
  ggplot(aes(x=Mes,y=AVG_GWETTOP,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_fill_hue(direction = -1) +
  theme_minimal()
```

En general, la humedad superficial de suelo media entre los casos positivos es inferior que entre los casos negativos. Esto es especialmente evidendente entre los meses de primavera y otoño. En los meses de invierno esto no está tan claro, incluso parece revertirse la tendencia.

1.4.2.4. WS10M

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=WS10M,alpha=WS10M,size=as.numeric(ifelse(WS10M<7.5,
  scale_color_gradientn(colours = rainbow(5,rev=T)) +
  facet_wrap(~month(date,label=TRUE))+
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de WS10M por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  guides(size="none",alpha="none")
```

Los valores más elevados de la velocidad del viento a 10m de altura se dan en las zonas costeras.

```
datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
  summarize(AVG_WS10M = mean(WS10M),
            Mes = `month(date, label = T)`) %>%
  ggplot(aes(x=Mes,y=AVG_WS10M,fill=fire)) +
  geom_col(position="dodge",alpha=0.8)+
  scale_fill_hue(direction = -1) +
  theme_minimal()
```

En todos los meses, los valores medios de la velocidad del viento a 10m sobre la superficie son significativamente mayores en las observaciones positivas.

1.4.2.5. WD10M

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=WD10M),size = 1.2, alpha = 0.6) +
  facet_wrap(~month(date,label=TRUE))+
  # scale_color_stepsn(colours = rainborainbpw(8,rev=T)) +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de WD10M por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  guides(color = guide_legend(override.aes = list(size = 5))) +
  scale_color_viridis_d(option="turbo")
```

```
# Quisiera tener la dirección mayoritaria por mes y si hay incendio o no
mode <- function(x) { names(which.max(table(x))) }

datos_mode_WD10M = datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
  summarize(mode_WD10M = mode(WD10M))

tibble(mes = datos_mode_WD10M[datos_mode_WD10M$fire==1,1],
       fire1 = datos_mode_WD10M[datos_mode_WD10M$fire==1,3],
       fire0 = datos_mode_WD10M[datos_mode_WD10M$fire==0,3])

# No parece aportar ninguna información relevante
```

1.4.2.6. PRECTOTCORR

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=PRECTOTCORR,size=PRECTOTCORR,alpha=PRECTOTCORR)) +
  facet_wrap(~month(date,label=TRUE)) +
  scale_color_gradientn(colours = rainbow(5,rev=F)) +
  guides(size="none") +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de PRECTOTCORR por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
```

```
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank()) +
  guides(alpha="none")
```

En la gran mayoría de las observaciones no se han observado precipitaciones. Se observan algunos valores positivos de precipitaciones distribuidos por el territorio Andaluz, alcanzándose los máximos en observaciones localizadas ne las coordilleras béticas.

```
datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
  summarize(AVG_PRECTOTCORR = mean(PRECTOTCORR),
            Mes = `month(date, label = T)`) %>%
  ggplot(aes(x=Mes,y=AVG_PRECTOTCORR,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_fill_hue(direction = -1) +
  theme_minimal()
```

Se observa una clara diferencia en la media mensual de precipitaciones en ambas clases en todos los meses, siendo mucho mayores entre las observaciones negativas. En los meses de verano esto sigue siendo cierto, si bien las diferencias se suavizan significativamente, ya que en ambos clases las precipitaciones son reducidas.

1.4.2.7. NDVI

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=NDVI),size = 1.2, alpha = 0.6) +
  facet_wrap(~month(date,label=TRUE)) +
  scale_color_gradientn(colours = rainbow(5,rev=T)) +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de NDVI por mes") +
  theme_minimal() +
  theme(axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

Los valores más elevados del NDVI se encuentran en el norte de la comunidad (Sierra Morena y Sierra de Aracena) y en la zona sur-centro. Los valores son especialmente

elevados en la Sierra de Grazalema y en las marismas del Guadalquivir y especialmente bajos en el este de la comunidad (Almería).

```
datos %>%
  st_drop_geometry() %>%
  group_by(month(date,label=T),fire) %>%
  summarize(AVG_NDVI = mean(NDVI),
            Mes = `month(date, label = T)`) %>%
  ggplot(aes(x=Mes,y=AVG_NDVI,fill=fire)) +
  geom_col(position="dodge",alpha=0.8) +
  scale_fill_hue(direction = -1) +
  theme_minimal()
  # theme(axis.text.x = element_text(angle = 45, vjust = 0.5, hjust=0.5))
```

El valor del NDVI disminuye en los meses de verano y alcanza su máximo en los meses de invierno. No se observa claramente una relación entre el NDVI y la clase de la observación en la muestra.

1.4.2.8. poblacion

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=poblacion,size = poblacion,alpha=poblacion/median(p
  scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de poblacion") +
  theme_minimal()

by(datos$poblacion,datos$fire,summary)
```

En la gran mayoría del territorio se observan niveles de población inferiores a 20.000 habitantes por municipio. Destacan las capitales de provincia por tener valores significativamente más elevados.

1.4.2.9. dens_poblacion

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dens_poblacion, size= dens_poblacion, alpha=dens_po
  scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  # scale_color_gradient(low="blue", high="red")+
  ggtitle("Distribución espacial de dens_poblacion") +
```

```
  theme_minimal()

by(datos$dens_poblacion,datos$fire,summary)
```

Comentario similar a la población. Mayor densidad de población en las zonas de costa y en las capitales de provincia.

1.4.2.10. pendiente

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=pendiente, size= pendiente, alpha=pendiente)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de pendiente") +
  theme_minimal()
```

1.4.2.11. elevacion

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=elevacion, size= elevacion, alpha=elevacion)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de elevacion") +
  theme_minimal()
```

1.4.2.12. curvatura

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=curvatura, alpha=curvatura)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de curvatura") +
  theme_minimal()
```

1.4.2.13. dist_carretera

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dist_carretera, alpha=dist_carretera)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de dist_carretera") +
  theme_minimal()
```

1.4.2.14. dist_electr

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dist_electr, alpha=dist_electr)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de dist_electr") +
  theme_minimal()
```

1.4.2.15. dist_camino

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dist_camino, alpha=dist_camino)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de dist_camino") +
  theme_minimal()
```

1.4.2.16. dist_sendero

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dist_sendero, alpha=dist_sendero)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de dist_sendero") +
  theme_minimal()
```

1.4.2.17. dist_poblacion

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dist_poblacion, alpha=dist_poblacion)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de dist_poblacion") +
  theme_minimal()
```

1.4.2.18. dist_ferrocarril

```
ggplot(data = prov) +
  geom_sf() +
  geom_sf(data = datos, aes(color=dist_ferrocarril, alpha=dist_ferrocarril)) +
    scale_color_gradientn(colours = rainbow(5,rev=T)) +
  guides(size="none",alpha="none") +
  ggtitle("Distribución espacial de dist_ferrocarril") +
  theme_minimal()
```

1.4.3. Análisis multivariantes

1.4.3.1. Variables numéricas

```
datos_numeric = datos %>%
  select(where(is.numeric)) %>%
  st_drop_geometry()

# Correlaciones ------------------------------------------------------------
R = cor(datos_numeric)
corrplot(R,method = "ellipse",type = "lower")
summary(R-diag(diag(R)))
# Las variables más correlacionadas en la muestra son T2M con RH2M (negativamente, -


# Gráfico de coordenadas paralelas -----------------------------------------
datos |>
  select(fire,where(is.numeric)) |>
  ggparcoord(columns=2:19,alphaLines=0.1,groupColumn = "fire") +
  xlab('') +
  ylab('') +
  scale_x_discrete(labels=colnames(datos_numeric)) +
  scale_color_hue(direction = -1) +
  # guides(color ="none")+
  theme_minimal() +
```

```
  ggtitle("Tipificación a normal estándar (por defecto)") +
  theme(plot.title = element_text(size = 9),
        axis.text.x = element_text(angle = 90))

# Boxplots ------------------------------------------------------------
boxplots <- map(names(datos_numeric), ~ ggplot(datos, aes(x = fire, y = .data[[.x]],f
                  geom_boxplot() +
                  scale_fill_hue(direction = -1) +
                  guides(fill="none") +
                  labs(title = paste(.x, "~ fire")))
ggarrange(plotlist = boxplots,ncol=6,nrow=3)

# PCA -----------------------------------------------------------------
pca <- princomp(datos_numeric,cor=T)
summary(pca)
# Se necesitan al menos 11 componentes principales para explicar el 80% de la varia
```

1.4.3.2. Variables categóricas

```
datos_categ <- datos %>%
  select(WD10M,orientacion,enp,uso_suelo) %>%
  st_drop_geometry()

# Histogramas ---------------------------------------------------------
histogramas <- map(names(datos_categ), ~ ggplot(datos, aes(x = .data[[.x]], fill = fi
                    geom_bar(position = "dodge", alpha = 0.8) +
                    scale_fill_hue(direction = - 1) +
                    theme_minimal() +
                    labs(title = .x)+theme(axis.text.x = element_text(size=7)))

ggarrange(plotlist = histogramas, ncol = 2, nrow = ceiling(length(histogramas)/2))
```

1.5. Modelos

```
# Librerías -----------------------------------------------------------
# Se cargan las librerías que se usarán en esta sección
library(tidyverse) # Manipulación de datos
library(sf) # Vector data
library(tidymodels) # Ecosistema para la construcción de modelos
library(akima) # Función interp
library(magrittr) # Operador %<>%
library(ggpubr) # Función ggarrange
library(knitr) # Función kable
```

```
# Carga de datos ------------------------------------------------------------
load("salidas_intermedias/datos_strat_depurados_geom_2024-05-03.RData")

# Agrupación clases uso_suelo -----------------------------------------------
# Nos quedamos con los 7 niveles del factor más frecuentes (clases 2 y 3)
datos <- datos |>
  mutate(uso_suelo = fct_lump(uso_suelo,
                              n = 7,
                              other_level= "Otro"))

# Funciones para la evaluación de modelos -----------------------------------
# Función para obtener las medidas de rendimiento de los modelos a partir de un obje
get_metrics <- function(pred) {
  list(
    res = tibble(
      roc_auc = pred |> roc_auc(truth = fire, .pred_0) |> pull(.estimate),
      accuracy = pred |> accuracy(truth = fire, .pred_class) |> pull(.estimate),
      recall = pred |> sensitivity(truth = fire, .pred_class,event_level="second") |>
      specificity = pred |> spec(truth = fire, .pred_class,event_level="second") |> p
      precision = pred |> precision(truth = fire, .pred_class,event_level="second") |
    conf_mat = pred |> conf_mat(truth = fire, .pred_class))
}

# Función para mostrar gráficamente los resultados del tuning de un modelo con dos p
tuning_plot = function(mod_res) {
  datos_metrics = mod_res %>%
    collect_metrics()

  plots = list()

  for (metric in unique(datos_metrics$.metric)) {

    datos = datos_metrics %>%
      filter(.metric==metric)

    # Interpolar los datos faltantes
    datos_interp <- interp(datos[[1]], datos[[2]], datos$mean)

    # Crear un nuevo dataframe con los datos interpolados
    datos_interp_df <- data.frame(
      expand.grid(x = datos_interp$x, y = datos_interp$y), z = as.vector(datos_interp

    # Crear el gráfico de mapa de calor con interpolación
    p = ggplot(datos_interp_df, aes(x = x, y = y, fill = z)) +
      geom_tile() +
      scale_fill_viridis_c(option = "turbo", name = NULL,na.value = "transparent")+
      labs(title = "",
```

```
            x = colnames(datos)[1],
            y = colnames(datos)[2],
            fill = metric) +
       theme_minimal()

    plots[[metric]] = p
  }
  ggarrange(plotlist = plots,
            labels=c("Accuracy","Specificy","ROC-AUC","Recall"),
            align = "hv")
}
```

1.5.1. Partición temporal entrenamiento / validación / test

```
set.seed(123)

splits = initial_validation_time_split(datos,
                                       prop=c(0.6,0.2))

training <- training(splits) %>%  st_drop_geometry()
val_set <- validation_set(splits) %>% st_drop_geometry()
test  <- testing(splits) %>% st_drop_geometry()
```

1.5.2. Regresión logística con penalización

```
# 1º Definimos el modelo:
lr_mod <-
  logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")

# 2º Creamos la receta
lr_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow","month")) %>%
  # step_holiday(date, holidays = holidays)%>%
  step_rm(date,cod_municipio,municipio) %>% # Se eliminan variables identificadoras
  step_dummy(all_nominal_predictors()) %>% # Se crean variables dummy para los fact
  step_lincomb() %>% # Elimina variablies con dependencia lineal exacta
  step_corr() %>% # Elimina variables con correlación superior a 0.9
  step_zv(all_predictors()) %>% # Eliminar variables con varianza nula
  step_normalize(all_predictors()) # Se normalizan todos los predictores

# 3º Creamos el workflow
lr_workflow <-
```

```
  workflow() %>%
  add_model(lr_mod) %>%
  add_recipe(lr_recipe)

# 4º Creamos el grid para los parámetros
lr_reg_grid <- expand_grid(penalty = 10^seq(-4, -1, length.out = 10),
                           mixture = seq(0,1,length.out=10))

# 5º Ajustamos el modelo
lr_res <-
  lr_workflow %>%
  tune_grid(val_set,
            grid = lr_reg_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# 6º Evaluación de modelos
tuning_plot(lr_res)

lr_res |>
  collect_metrics() |>
  group_by(.metric)|>
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) |>
  summarise(max = max(mean),min=min(mean))


# 7º Selección del mejor modelo
lr_best <-
  lr_res %>%
  select_best(metric="accuracy")
lr_best

# Extraer coeficientes
lr_workflow %>%
  finalize_workflow(lr_best) %>%
  fit(training) %>%
  extract_fit_parsnip() %>%
  tidy() %>%
  print(n=100)
```

1.5.3. Regresión logística con penalización + PCA

```
# 1º Creamos el modelo
lr_pca_mod <-
```

```
  logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet")


# 2º Creamos la receta
lr_pca_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow","month")) %>%
  # step_holiday(date, holidays = holidays)%>%
  step_rm(date,cod_municipio,municipio) %>% # Se eliminan variables identificadoras
  step_dummy(all_nominal_predictors()) %>% # Se crean variables dummy para los fact
  step_lincomb() %>% # Elimina variablies con dependencia lineal exacta
  step_corr() %>% # Elimina variables con correlación superior a 0.9
  step_zv(all_predictors()) %>% # Eliminar variables con varianza nula
  step_normalize(all_predictors()) %>% # Se normalizan todos los predictores
  step_pca(all_numeric_predictors(),num_comp = tune())

# 3º Creamos el workflow
lr_pca_workflow <-
  workflow() %>%
  add_model(lr_pca_mod) %>%
  add_recipe(lr_pca_recipe)

# 4º Creamos el grid para los parámetros
lr_pca_reg_grid <- expand_grid(penalty = 10^seq(-4, -1, length.out = 10),
                               mixture = seq(0,1,length.out=10),
                               num_comp = c(20,25,30,35,40,45,50))

# 5º Ajustamos el modelo
lr_pca_res <-
  lr_pca_workflow %>%
  tune_grid(val_set,
            grid = lr_pca_reg_grid,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# 6º Evalación modelos
lr_pca_tuning = lr_pca_res |>
  collect_metrics() |>
  group_by(.metric)|>
  summarise(max = max(mean),min=min(mean))

# 7º Selección del mejor modelo
lr_pca_best <-
  lr_pca_res %>%
  select_best(metric="accuracy")
lr_pca_best
```

1.5.4. Árboles de decisón

```
# 1º Creamos el modelo
dt_mod <-
  decision_tree(cost_complexity = tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification")

# 2º Creamos la receta
dt_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow", "month")) %>%
  # step_holiday(date)%>%
  step_rm(date, cod_municipio, municipio)

# 3º Creamos el workflow
dt_workflow <-
  workflow() %>%
  add_model(dt_mod) %>%
  add_recipe(dt_recipe)

# 4º Se ajusta el modelo
set.seed(345)
dt_res <-
  dt_workflow %>%
  tune_grid(val_set,
            grid = 10,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# 5º Se evalúan los modelos obtenidos
dt_res |>
  collect_metrics() |>
  group_by(.metric)|>
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) |>
  summarise(max = max(mean),min=min(mean))

# Gráfico del ajuste
  dt_res %>%
  collect_metrics() %>%
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) |>
  ggplot(aes(x = cost_complexity, y = mean,col=.metric)) +
  geom_point() +
```

```
  geom_line() +
  ylab("") +
  scale_x_log10(labels = scales::label_number())+
  theme_minimal()

# 6º Selección del mejor modelo
dt_best <- dt_res |>
  select_best(metric = "accuracy")
dt_best
```

1.5.5. Bosques aleatorios

```
# Detectar el número de núcleos para trabajar en paralelo
cores <- parallel::detectCores()
cores

# Construimos el modelo, especificando el número de núcleos a usar en la computación

# ETAPA 1: fijado mtry=4, se ajusta min_n
# -----------------

# 1º Construir el modelo
rf_mod1 <-
  rand_forest(mtry = 4, min_n = tune(), trees = 1000) %>%
  set_engine("ranger", num.threads = cores) %>%
  set_mode("classification")

# 2º Construir la receta con el preprocesamiento
rf_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow", "month")) %>%
  # step_holiday(date)%>%
  step_rm(date, cod_municipio, municipio)
# No normalizamos en este caso pues no es necesario

# 3º Ensamblar todo con workflow
rf_workflow1 <-
  workflow() %>%
  add_model(rf_mod1) %>%
  add_recipe(rf_recipe)

# 4º Train and tune
set.seed(345)

rf_res1 <-
  rf_workflow %>%
```

```
  tune_grid(val_set,
            grid = expand_grid(min_n = seq(1000,2500,100)),
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# Resultados del tuning

rf_tuning1 <- rf_res1 |>
  collect_metrics() |>
  group_by(.metric)|>
  summarise(max = max(mean),min=min(mean))

rf_tuning1

# plot

rf_plot1 <-
  rf_res1 %>%
  collect_metrics() %>%
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) %>%
  ggplot(aes(x = min_n, y = mean,col=.metric)) +
  geom_point() +
  geom_line() +
  ylab("") +
  theme_minimal()+
  labs(title = "Etapa 1\nFijado mtry = 4, se ajusta min_n")


# Mejor modelo
rf_best1 <-
  rf_res1 %>%
  select_best(metric = "spec")

rf_best1

rf_metrics1 <- rf_res1 |>
  collect_predictions(parameters = rf_best1) |>
  get_metrics()

rf_metrics1


# ETAPA 2: fijado min_n de la etapa anterior, se ajusta mtry
# -----------------

# 1º Se construye el modelo
```

```r
rf_mod2 <-
  rand_forest(mtry = tune(), min_n = rf_best1$min_n, trees = 1000) %>%
  set_engine("ranger", num.threads = cores) %>%
  set_mode("classification")

# 2º Se usa la misma receta que antes

# 3º Ensamblar todo con workflow
rf_workflow2 <-
  workflow() %>%
  add_model(rf_mod2) %>%
  add_recipe(rf_recipe)

# 4º Train and tune
set.seed(345)

rf_res2 <-
  rf_workflow2 %>%
  tune_grid(val_set,
            grid = expand_grid(mtry = 1:10),
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))


# Resultados del tuning

rf_tuning2 <- rf_res2 |>
  collect_metrics() |>
  group_by(.metric)|>
  summarise(max = max(mean),min=min(mean))

rf_tuning2

#plot
rf_plot2 <-
  rf_res2 %>%
  collect_metrics() %>%
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) %>%
  ggplot(aes(x = mtry, y = mean,col=.metric)) +
  geom_point() +
  geom_line() +
  ylab("") +
  theme_minimal()+
  labs(title = paste0("Etapa 2\nFijado min_n = ", rf_best1$min_n, " se ajusta mtry"))

# Mejor modelo
```

```
rf_best2 <-
  rf_res2 %>%
  select_best(metric = "spec")
rf_best2

rf_metrics2 <- rf_res2 |>
  collect_predictions(parameters = rf_best2) |>
  get_metrics()

rf_metrics2


# ---------

# Plots
ggarrange(rf_plot1,rf_plot2,nrow=1,common.legend = T,legend = "bottom")
```

1.5.6. k-Nearest Neighbours

```
# 1º Definimos el modelo:
knn_mod <-
  nearest_neighbor(neighbors = tune()) %>%
  set_engine("kknn") %>%
  set_mode("classification")

# 2º Creamos la receta
knn_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow","month")) %>%
  # step_holiday(date, holidays = holidays)%>%
  step_rm(date,cod_municipio,municipio) %>% # Se eliminan variables identificadoras
  step_dummy(all_nominal_predictors()) %>% # Se crean variables dummy para los fact
  step_lincomb() %>% # Elimina variablies con dependencia lineal exacta
  step_corr() %>% # Elimina variables con correlación superior a 0.9
  step_zv(all_predictors()) %>% # Eliminar variables con varianza nula
  step_normalize(all_predictors()) # Se normalizan todos los predictores

# 3º Creamos el workflow
knn_workflow <-
  workflow() %>%
  add_model(knn_mod) %>%
  add_recipe(knn_recipe)

# 4º Train and tune
set.seed(345)
knn_res <-
```

```
  knn_workflow %>%
  # fit(training)
  tune_grid(val_set,
            grid = expand_grid(neighbors = c(1,10,25,seq(25,400,25))),
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# 5º Evaluació de los modelos
knn_res |>
  collect_metrics() |>
  group_by(.metric)|>
  summarise(max = max(mean),min=min(mean))

knn_res %>%
  collect_metrics() %>%
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) %>%
  # filter(.metric == "accuracy") %>%
  ggplot(aes(x = neighbors, y = mean,col=.metric)) +
  geom_point() +
  geom_line() +
  ylab("") +
  theme_minimal()

# 6º Selección del mejor modelo
knn_best <- knn_res |>
  select_best(metric = "accuracy")
knn_best
```

1.5.7. SVM lineal

```
# 1º Construir el modelo
svm_mod <-
  svm_linear(cost = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

# 2º Construir la receta con el preprocesamiento
svm_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow", "month")) %>%
  step_rm(date, cod_municipio, municipio) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())
```

```
# 3º Ensamblar todo con workflow
svm_workflow <-
  workflow() %>%
  add_model(svm_mod) %>%
  add_recipe(svm_recipe)

# 4º Train and tune
set.seed(345)
svm_res <-
  svm_workflow %>%
  tune_grid(val_set,
            grid = 15,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# 5º Evaluación de resultados del tuning
svm_res |>
  collect_metrics() |>
  group_by(.metric)|>
  summarise(max = max(mean),min=min(mean))

svm_plot <-
  svm_res %>%
  collect_metrics() %>%
  mutate(.metric = ifelse(.metric == "recall","spec",
                          ifelse(.metric == "spec","recall",
                                 .metric))) %>%
  # filter(.metric == "accuracy") %>%
  ggplot(aes(x = cost, y = mean,col=.metric)) +
  geom_point() +
  geom_line() +
  ylab("") +
  theme_minimal()
svm_plot

# 6º Selección del mejor modelo
svm_best <-
  svm_res %>%
  select_best(metric="accuracy")
svm_best
```

1.5.8. SVM radial

```
# 1º Construir el modelo
svm_rbf_mod <-
```

```
  svm_rbf(cost = tune(),rbf_sigma = tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

# 2º Construir la receta con el preprocesamiento
svm_rbf_recipe <-
  recipe(fire ~ ., data = training) %>%
  step_date(date,features = c("dow", "month")) %>%
  step_rm(date, cod_municipio, municipio) %>%
  step_dummy(all_nominal_predictors()) %>%
  step_lincomb() %>% # Elimina variablies con dependencia lineal exacta
  step_corr() %>% # Elimina variables con correlación superior a 0.9
  step_zv(all_predictors()) %>%
  step_normalize(all_predictors())

# 3º Ensamblar todo con workflow
svm_rbf_workflow <-
  workflow() %>%
  add_model(svm_rbf_mod) %>%
  add_recipe(svm_rbf_recipe)

# 4º Train and tune
set.seed(345)
svm_rbf_res <-
  svm_rbf_workflow %>%
  tune_grid(val_set,
            grid = 8,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(accuracy,roc_auc,recall,spec))

# 5º Evaluación de resultados del tuning
svm_rbf_res |>
  collect_metrics() |>
  group_by(.metric)|>
  summarise(max = max(mean),min=min(mean))

# 6º Selección del mejor modelo
svm_rbf_best <-
  svm_rbf_res %>%
  select_best(metric="accuracy")

svm_rbf_best
```

1.5.9. Comparación

```
models = tibble(model_name = c("lr","lr_pca","dt","rf","svm_linear","svm_rbf","knn"),
                models_tune = list(lr_res,lr_pca_res,dt_res,rf_res2,svm_res,svm_rbf_r
                models_workflow = list(lr_workflow,lr_pca_workflow,dt_workflow,rf_wor

# save(models, file="salidas_intermedias/all_models.RData")

load("salidas_intermedias/all_models.RData")
models = models %>%
  mutate(best_tuning = map(models_tune,function(x) select_best(x,metric = "accuracy")
         best_metrics = map2(models_tune,
                             best_tuning,
         roc = map2(models_tune,
                    best_tuning,
                    ~ collect_predictions(.x,parameters = .y) %>%
)

# metricas
metrics = models %>%
  select(model_name,best_metrics) %>%
  unnest(best_metrics)
kable(metrics,digits=3)

# curva roc
metrics %>%
  pivot_longer(cols = c(roc_auc, accuracy, recall, specificity, precision),
               names_to = "metric") %>%
  ggplot(aes(x = metric, y = value, group = model_name)) +
  geom_line(aes(col = model_name),size=1) +
  geom_point(aes(col = model_name),size=2.3) +
  scale_color_viridis_d(option="turbo") +
  geom_vline(xintercept=1:5, linetype="dotted") +
  labs(col = "Modelo", title = "Métricas sobre validación") +
  theme_minimal() +
  theme(axis.line.x = element_line(color="black", size = 1),
        axis.line.y = element_line(color="black", size = 1))

# plot medidas
models %>% select(model_name,roc) %>% unnest(roc) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model_name)) +
  geom_path(lwd = 1, alpha = 0.7) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option="turbo") +
  labs(color="Modelo")+
  # scale_color_viridis_d(option = "turbo",name="Modelo") +
```

```
  theme_minimal() +
  theme(axis.line.x = element_line(color="black", size = 1),
        axis.line.y = element_line(color="black", size = 1))+
  ggtitle("Curva ROC en validación")
```

Capítulo 2

Test

Se unen los conjuntos training y validation para entrenar el modelo final

```
set.seed(345)
models <- models %>%
  mutate(final_workflow = map2(models_workflow, best_tuning,finalize_workflow),
         last_fit = map(final_workflow, function(x) last_fit(x,splits,add_validation_

models = models %>%
  mutate(test_metrics = map(last_fit,
                            ~collect_predictions(.x) %>%
                              extract2(1)), # Para extraer solo las medidas
         test_roc = map(last_fit,
                        ~collect_predictions(.x) %>%

# save(models, file="salidas_intermedias/all_models_test.RData")

# metricas en test
test_metrics = models %>% select(model_name,test_metrics) %>% unnest(test_metrics)
kable(test_metrics,digits=3)

# plot
test_metrics %>%
  pivot_longer(cols = c(roc_auc, accuracy, recall, specificity, precision),
               names_to = "metric") %>%
  ggplot(aes(x = metric, y = value, group = model_name)) +
  geom_line(aes(col = model_name),size=1) +
  geom_point(aes(col = model_name),size=2.3) +
  scale_color_viridis_d(option="turbo") +
  geom_vline(xintercept=1:5, linetype="dotted") +
  labs(col = "Modelo", title = "Métricas sobre test") +
  theme_minimal() +
  theme(axis.line.x = element_line(color="black", size = 1),
        axis.line.y = element_line(color="black", size = 1))
```

```r
# roc
models %>% select(model_name,test_roc) %>% unnest(test_roc) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity, col = model_name)) +
  geom_path(lwd = 1, alpha = 0.7) +
  geom_abline(lty = 3) +
  coord_equal() +
  scale_color_viridis_d(option="turbo") +
  labs(color="Modelo")+
  # scale_color_viridis_d(option = "turbo",name="Modelo") +
  theme_minimal() +
  theme(axis.line.x = element_line(color="black", size = 1),
        axis.line.y = element_line(color="black", size = 1))+
  ggtitle("Curva ROC en test")
```

Para calcular la medida de importancia de las variables en bosque aleatorio. No se hace desde el principio para que la computación sea más rápida.

```r
# Se hace el last_fit manualmente:
cores = 8

# last model
last_rf_mod <- rand_forest(mtry = rf_best2$mtry, min_n = rf_best1$min_n, trees = 1000
  set_engine("ranger", num.threads = cores,importance="impurity") %>%
  set_mode("classification")

# last workflow
last_rf_workflow <-
  rf_workflow2 %>%
  update_model(last_rf_mod)

# last fit
set.seed(345)
last_rf_fit <-
  last_rf_workflow %>%
  last_fit(splits,
           add_validation_set = T)
# VIP
last_rf_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features = 50,aesthetics = list(fill="lightblue")) +
  theme_minimal()
```

2.1. Casos de interés

```r
# Librerías ------------------------------------------------------------------
# Se cargan las librerías que se usarán en esta sección
library(tidyverse) # Manipulación de datos
library(sf) # Vector data
library(tidymodels) # Ecosistema para la construcción de modelos
library(ggpubr) # Función ggarrange
library(knitr) # Función kable
library(skimr) # Función skim
library(terra)
```