

Trabajo FID

El dataset utilizado para todas las técnicas ha sido: https://www.kaggle.com/imdevskp/corona-virus-report?select=day_wise.csv

- Alcance del proyecto: Análisis avanzado e investigación BIGML.
- GitHub: <https://github.com/jesusmonda/fid>

Miembros del equipo:

- Roberto Hermoso Núñez (robhernun)
- Pablo García Barco (pabgarbar)
- María del Carmen Arenas Zayas (mararezay)
- Jesús Monda Caña (jesmoncaa, jmonda)

Reparto de tareas

Tareas	Personas implicadas
Preprocesamiento	Jesús y Roberto
Visualización	Roberto y Maricarmen
Árbol de decisión	Jesús y Roberto
Naive Bayes	Roberto
Maquinas de vectores de soporte	Roberto
Validación cruzada	Pablo
KMeans	Maricarmen
Principal component analysis	Maricarmen y Jesús
DBSCAN	Jesús
Heatmaps	Maricarmen
BIGML	Roberto

Preprocesamiento

Lo primero es necesario hacer un preprocesario. El dataset original se encuentra en la carpeta “Script”, el primer paso es normalizar las fechas, y después añadir una columna nueva indicando la estación del año en la que se encuentra cada dato. Para poder normalizar los días tuvimos que encontrar cuál es la fecha mas antigua, por cada iteración se calculó la diferencia de días de la fecha de cada dato con la fecha inicial, además de establecer a que estación pertenece. Esto último se hizo mediante un simple comparador de fechas.

Primero cargamos los datos

```
day_wise <- read.delim('script/day_wise.csv', sep=",", head = TRUE)
```

Después realizamos una función para determinar a que estación del año pertenece el día que le estemos indicando

```
getSeason <- function(DATES) {  
  WS <- as.Date("2012-12-21", format = "%Y-%m-%d") # Winter Solstice  
  SE <- as.Date("2012-3-21", format = "%Y-%m-%d") # Spring Equinox  
  SS <- as.Date("2012-6-21", format = "%Y-%m-%d") # Summer Solstice  
  FE <- as.Date("2012-9-23", format = "%Y-%m-%d") # Fall Equinox  
  
  # Convert dates from any year to 2012 dates  
  d <- as.Date(strftime(DATES, format="%Y-%m-%d"))  
  
  ifelse (d >= WS | d < SE, "Winter",  
    ifelse (d >= SE & d < SS, "Spring",
```

```
      ifelse (d >= SS & d < FE, "Summer", "Autumn"))))
}
```

Hacemos una copia del dataset

```
day_wise_procesado <- day_wise
day_wise_procesado$Season = rep(1,nrow(day_wise))
```

Y hallamos la fecha mas antigua, en este caso, se trata del 22 de enero de 2020

```
strDates <- day_wise[[1]]
dates <- as.Date(strDates, "%Y-%m-%d")
for(row in 1:nrow(day_wise)){
  date<-dates[row]
  if(row==1){
    earlies_date=date
  }else if(earlies_date>date){
    earlies_date<-date
  }
}
print(earlies_date)
```

```
## [1] "2020-01-22"
```

Después realizamos los cambios correspondientes al dataset

```
for (row in 1:nrow(day_wise)){
  data<-day_wise[row,]
  data$Season=getSeason(data$Date)
  data$Date<- as.Date(data$Date, "%Y-%m-%d") - earlies_date
  day_wise_procesado[row,]<-data
}
```

Y por último guardamos un fichero con estos datos para evitar tener que hacer el procesamiento si eliminamos las variables del proyecto.

```
write.csv(day_wise_procesado,"datos_nuevos_casos/regresion_train.csv", row.names = FALSE)
```

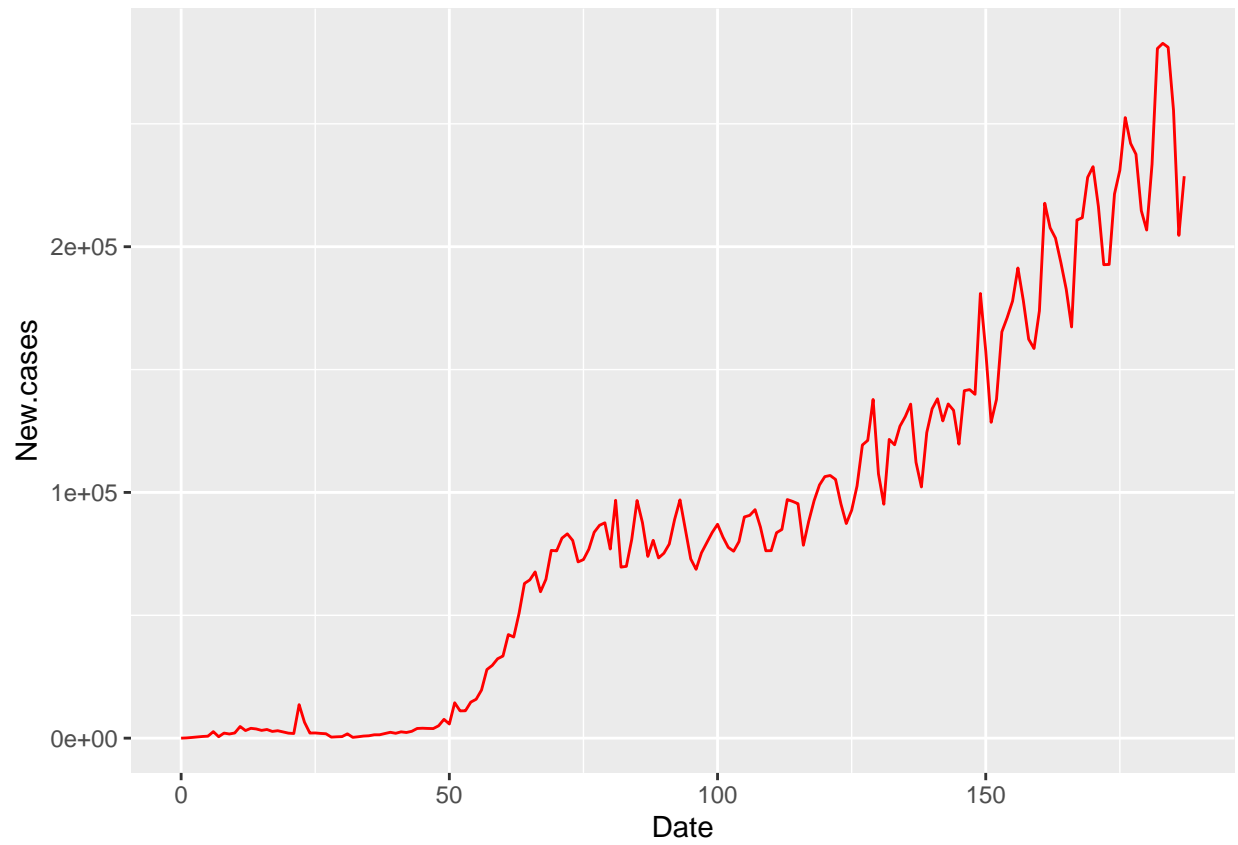
Visualización

```
library(ggplot2)
```

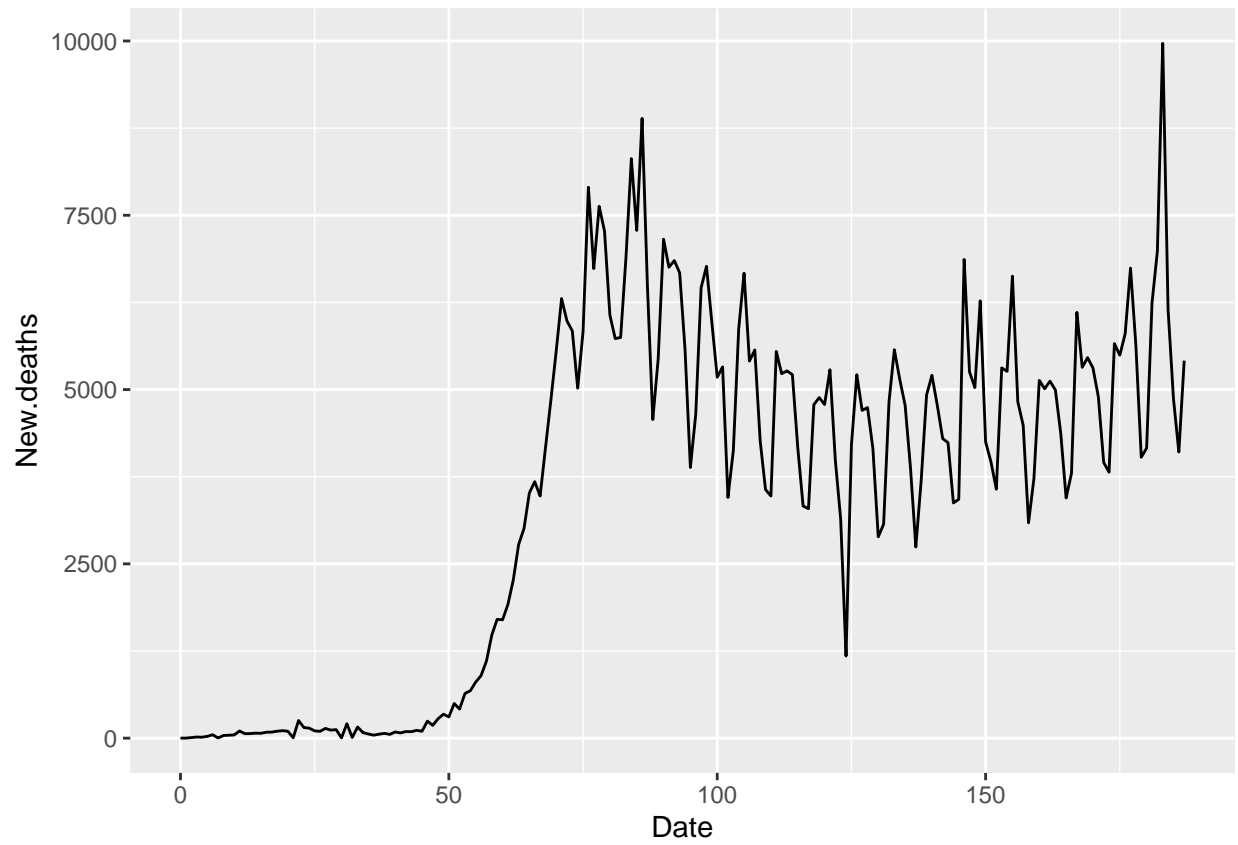
Cargamos y visualizamos los datos. Vamos a visualizar los nuevos casos, nuevos recuperados y nuevas muertes diarias.

```
data <- read.delim('datos_nuevos_casos/regresion_train.csv', sep=",", head = TRUE)

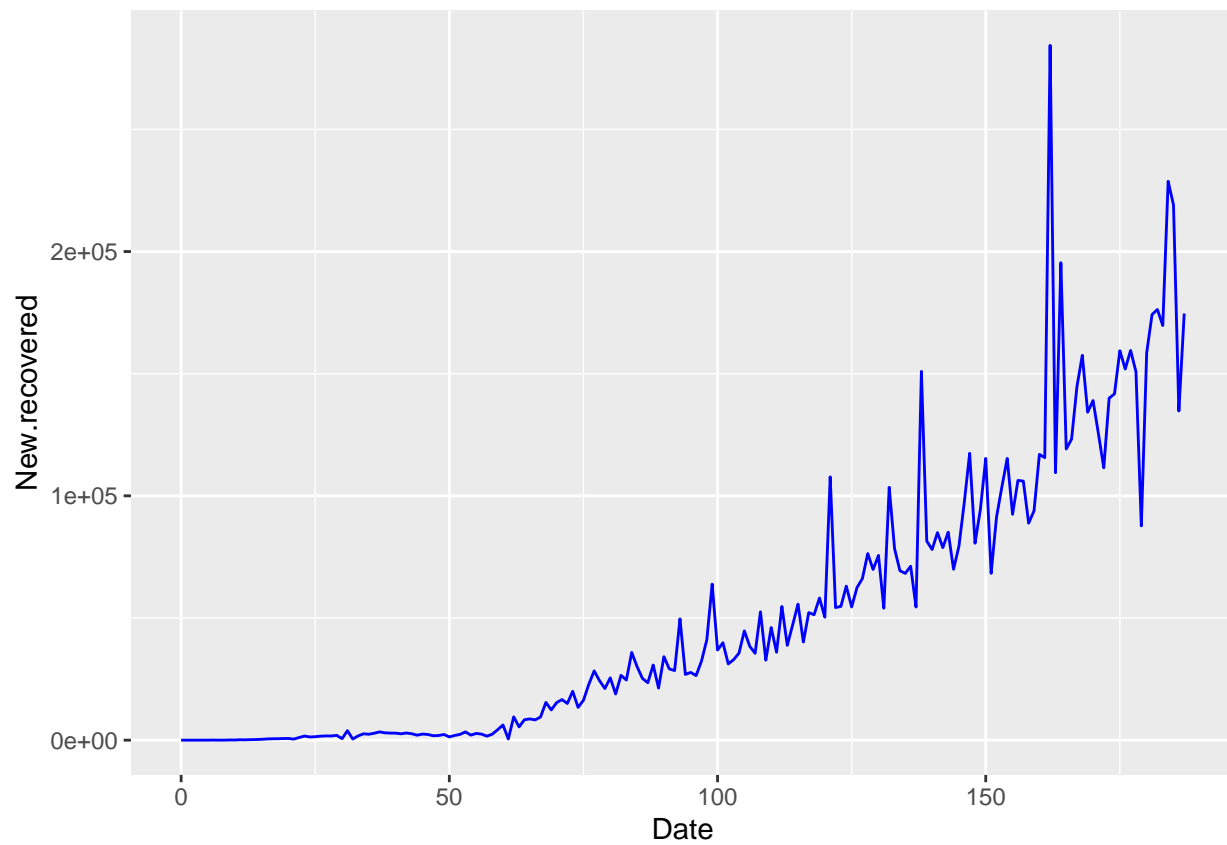
ggplot(data, aes(Date, New.cases)) + geom_line(color="red")
```



```
ggplot(data, aes(Date, New.deaths)) + geom_line(color="black")
```

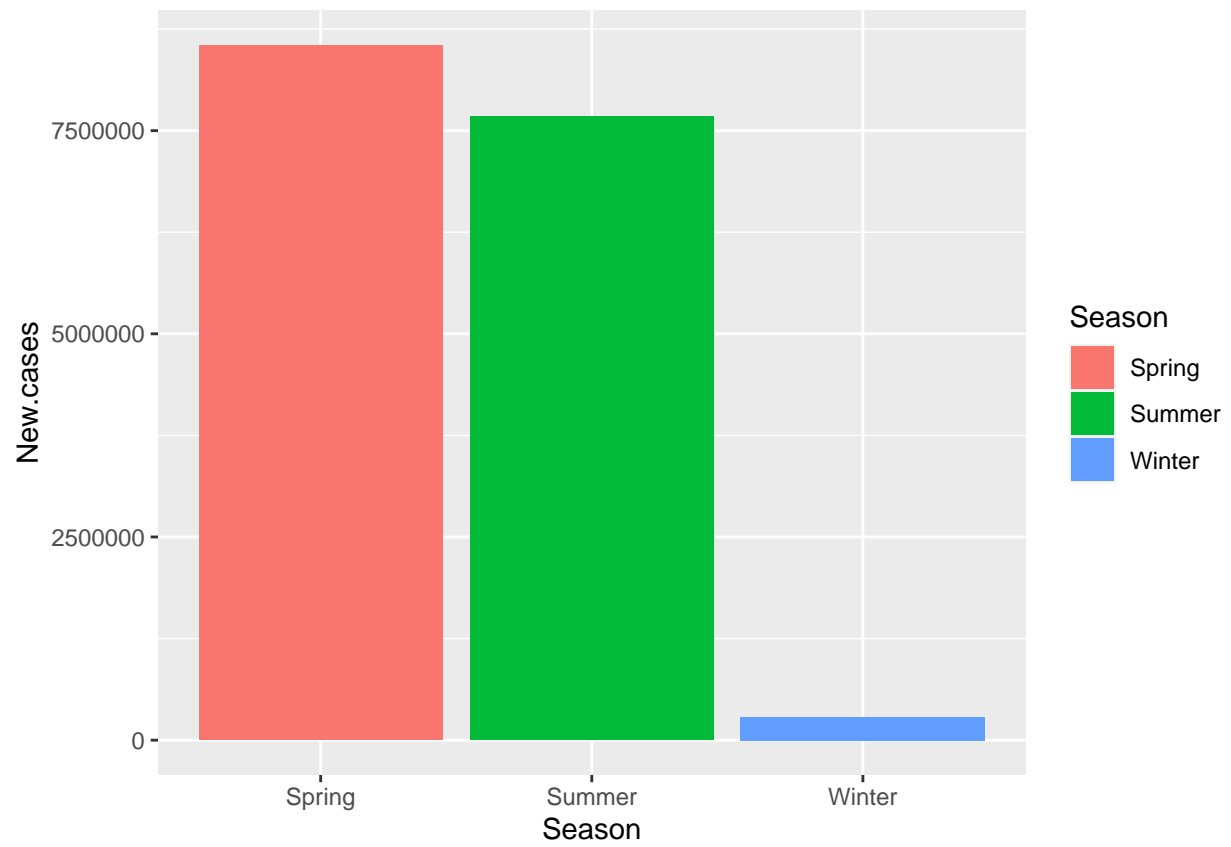


```
ggplot(data, aes(Date, New.recovered)) + geom_line(color="blue")
```

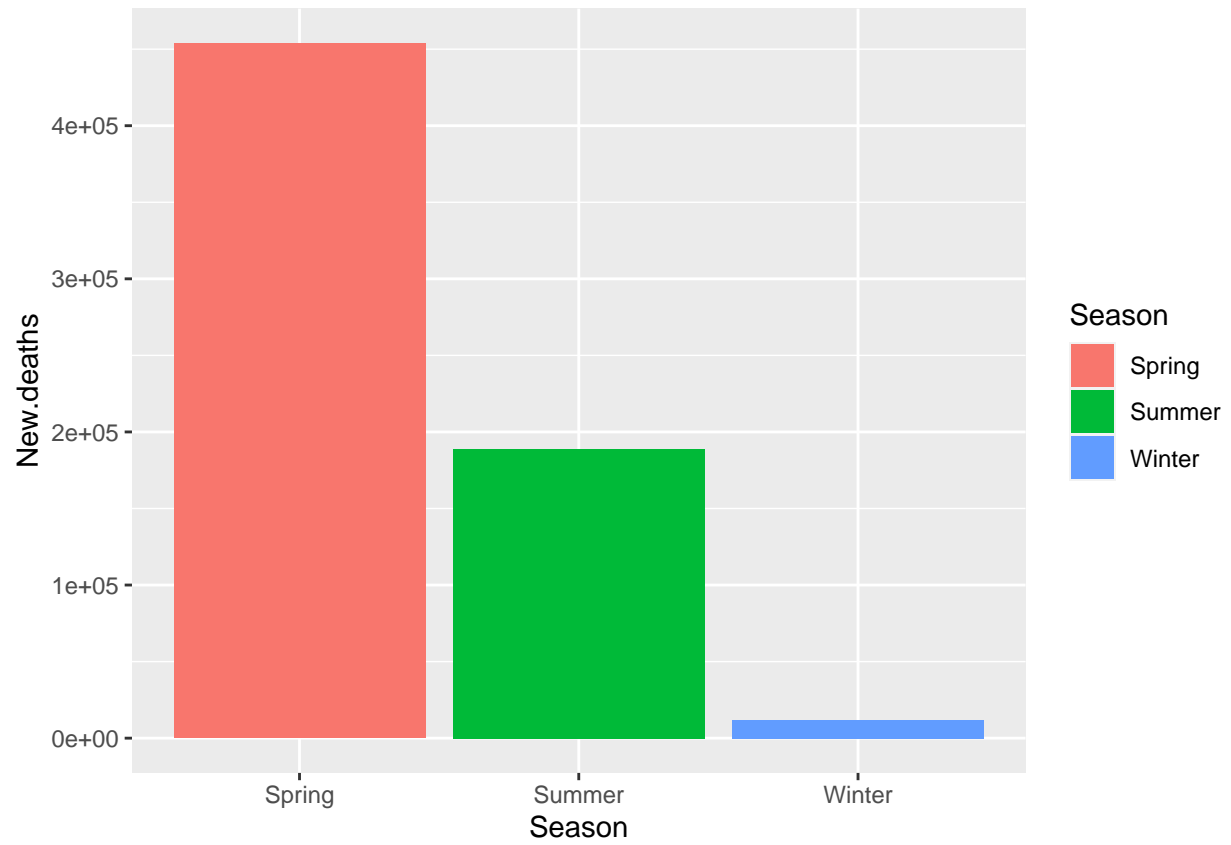


Y ahora mostraremos la densidad estos tres atributos por cada estación.

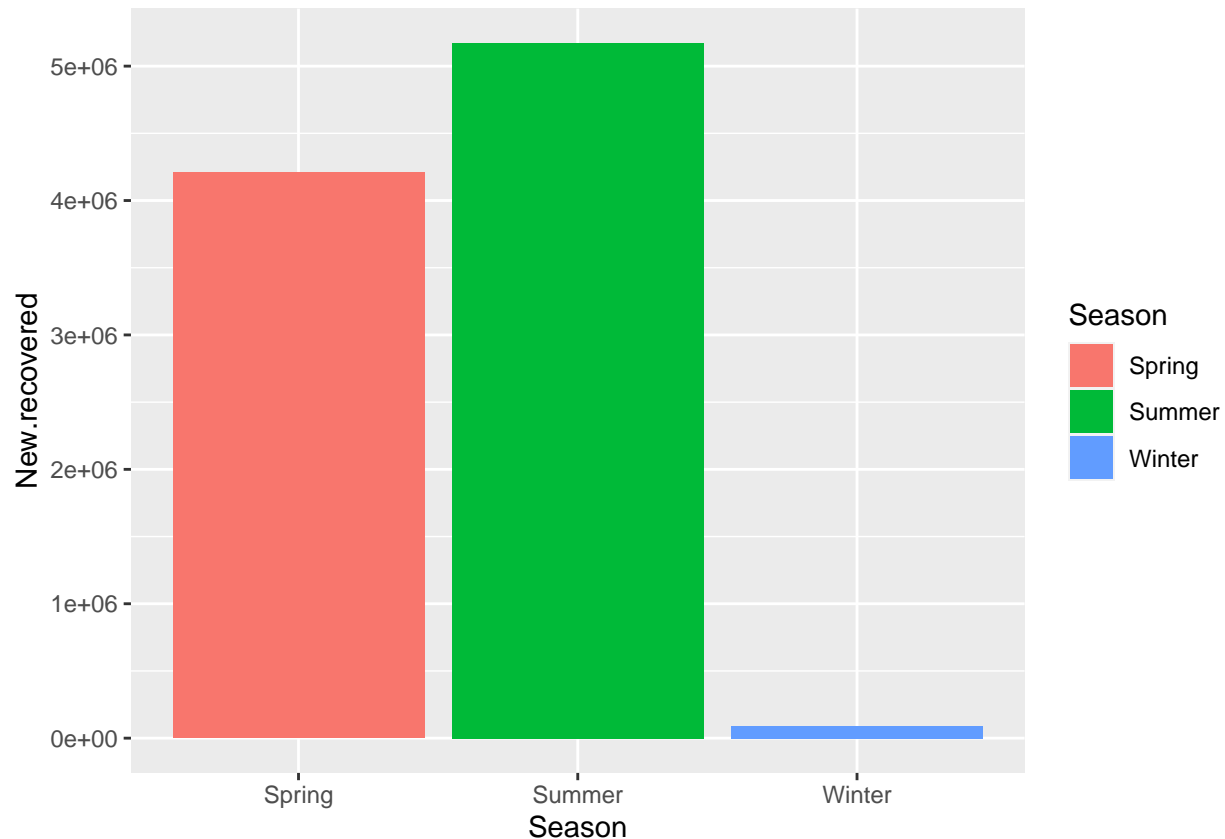
```
ggplot(data, aes(Season, New.cases, fill=Season)) + geom_bar(stat="identity")
```



```
ggplot(data, aes(Season, New.deaths, fill=Season)) + geom_bar(stat="identity")
```



```
ggplot(data, aes(Season, New.recovered, fill=Season)) + geom_bar(stat="identity")
```



Como podemos ver, hay muchos mas casos y muertes diarias en primavera, principalmente porque los datos empiezan a partir de enero y termina en julio, por lo que la mayoría de los datos se encuentran en primavera. Por otro lado, hay mas recuperados diarios en verano.

Predicción supervisada

Para estas predicciones hemos usado un dataset extraido de kaggle (https://www.kaggle.com/imdevskp/corona-virus-report?select=day_wise.csv) que muestra los datos del covid a nivel mundial divididos por días. Trataremos de predecir la estación del año a la que pertenece el dato, teniendo en cuenta los casos, las muertes y las recuperaciones diarias.

Clasificación mediante árbol

Los árboles de clasificación es un tipo de modelo de predicción que trata de predecir una variable en función de diversas variables de entrada. El árbol de clasificación lo componen una serie de nodos internos y externos así como los arcos que unen los nodos. Los nodos se les conocen como hojas del árbol y se marcan con una case o una distribución de probabilidad sobre las clases.

Instalamos pues, el paquete rattle para poder hacer la predicción mediante árbol

Cargamos el paquete

```
library(rpart)
library(rpart.plot)
library(caret)
```

```
## Loading required package: lattice
```

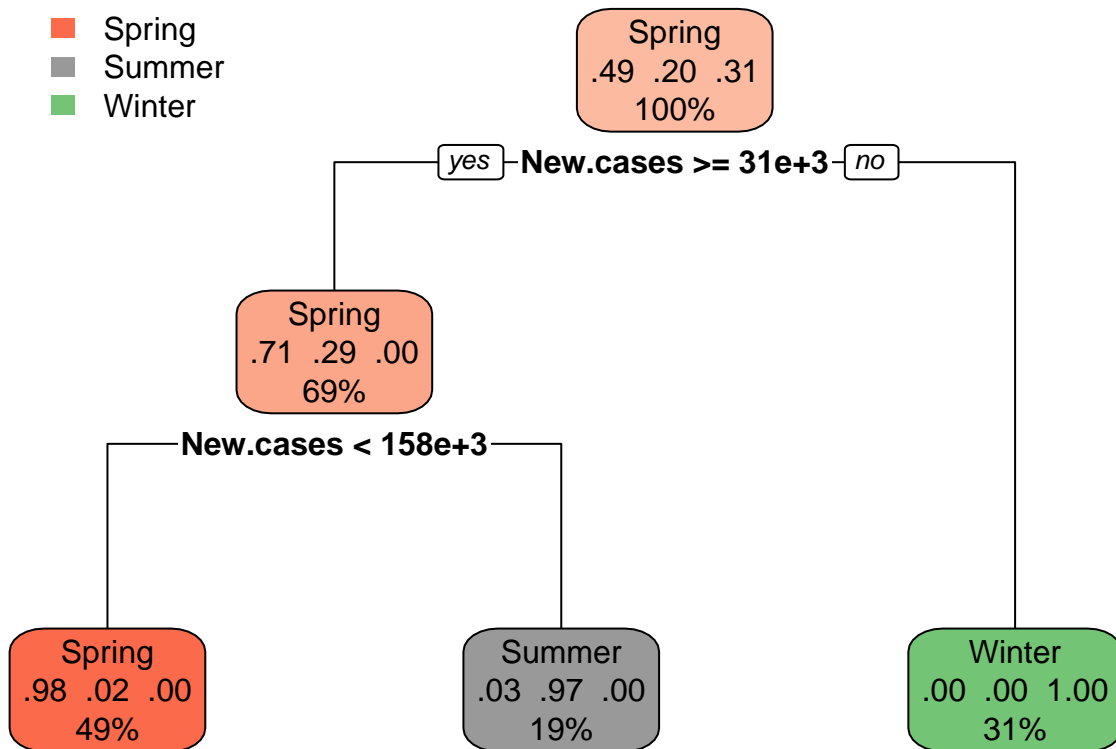


```
library(caTools)
```

```
data_arbol <- read.delim('datos_nuevos_casos/regresion_train.csv', sep=",", head = TRUE)
```

Ahora vamos a generar y a dibujar el árbol.

```
tree <- rpart(Season ~ New.cases + New.deaths + New.recovered, data_arbol, method="class")
rpart.plot(tree)
```



Como podemos ver en el árbol, pese a que estemos tratando de predecir mediante 3 variables, el árbol ha seleccionado la variable que considera mas importante para dividir el árbol, en este caso ha seleccionado la variable de “New Cases”.

Y por último hacer validacion cruzada con arbol de decision para probar su eficacia.

```
set.seed(1234)
split <- sample.split(data_arbol$Season, SplitRatio = 0.80)
training_set <- subset(data_arbol, split == TRUE)
test_set <- subset(data_arbol, split == FALSE)
```

```
table(training_set$Season)
```

```
##
## Spring Summer Winter
##      74      30      47
```

```
table(test_set$Season)
```

```
##
## Spring Summer Winter
```

```
##      18      7      12
folds <- createFolds(training_set$Season, k = 10)

cvDecisionTree <- lapply(folds, function(x){
  training_fold <- training_set[-x, ]
  test_fold <- training_set[x, ]
  clasificador <- rpart(Season ~ ., data = training_fold)
  y_pred <- predict(clasificador, newdata = test_fold, type = 'class')
  cm <- table(test_fold$Season, y_pred)
  precision <- (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])
  return(precision)
})
precisionDecisionTree <- mean(as.numeric(cvDecisionTree))
precisionDecisionTree
```

```
## [1] 0.9909091
```

Como podemos ver, la precisión es de un 99.09%

Fuente: <https://rpubs.com/rdelgado/405322>

Clasificación mediante Naive Bayes

Fuentes: <https://fervilber.github.io/Aprendizaje-supervisado-en-R/ingenio.html>, https://rpubs.com/riazakhan94/naive_bayes_classifier_e1071

Naive Bayes es un modelo para predecir por probabilidad Bayesiana, de tal forma que clasifica el resultado en función de variables que a priori son independientes entre sí. La formula de Bayes es:

$$P(B|A) = \frac{P(A|B) * P(B)}{P(A)}$$

Donde:

1. $P(A)$ es la probabilidad de que A sea cierto
2. $P(B)$ es la probabilidad de que B sea cierto
3. $P(A|B)$ es la probabilidad de que A sea cierto en función de B
4. $P(B|A)$ es la probabilidad de que B sea cierto en función de A

Teniendo esto en cuenta, procedemos a la realización del modelo usando Naive Bayes

Primero instalarnos el paquete naivebayes, además es necesario usar Rtools (<https://cran.r-project.org/bin/windows/Rtools/>)

Después cargamos el paquete

```
library(naivebayes)
```

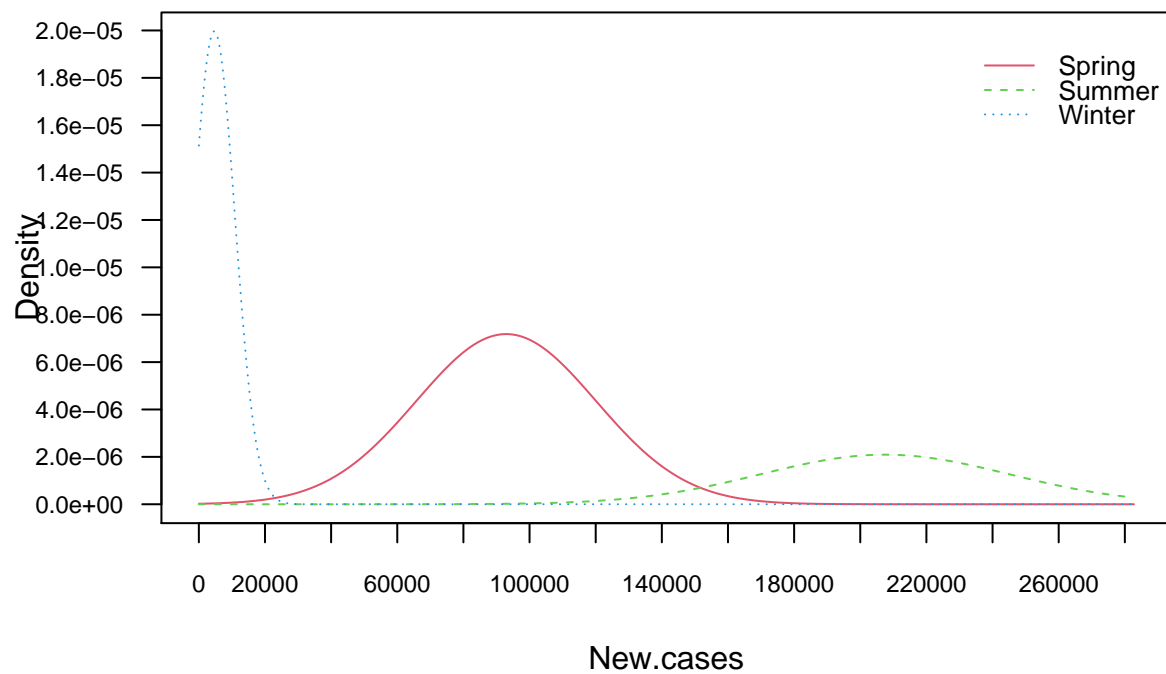
```
## naivebayes 0.9.7 loaded
```

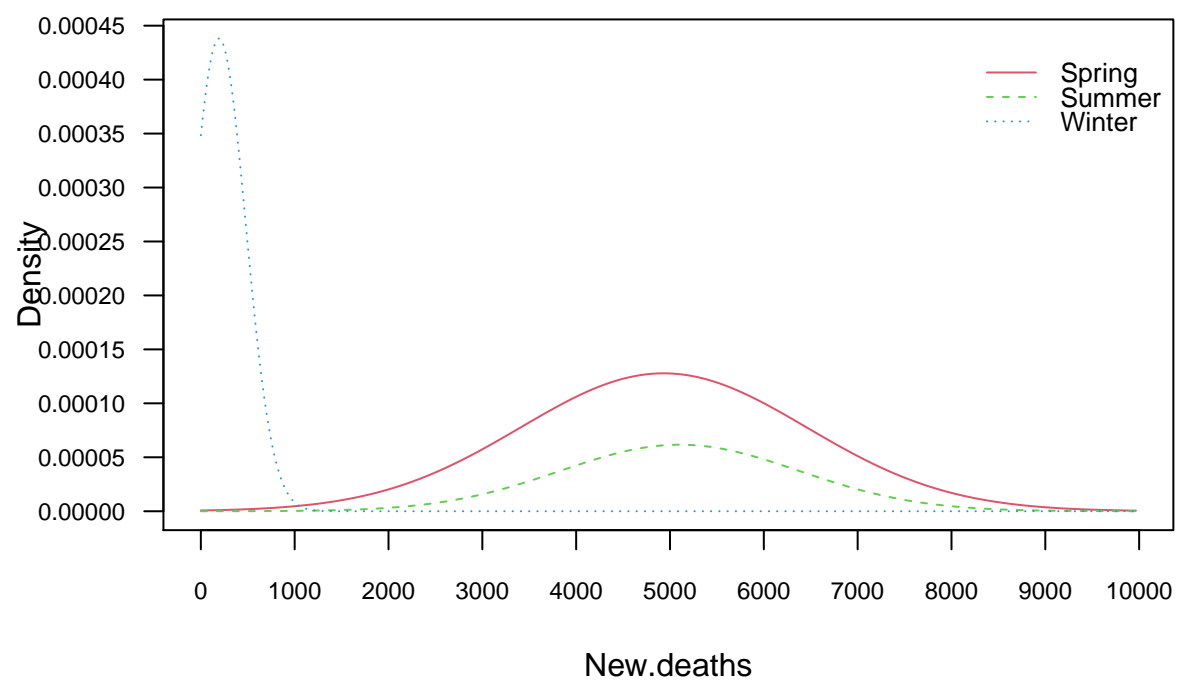
Cargamos los datos

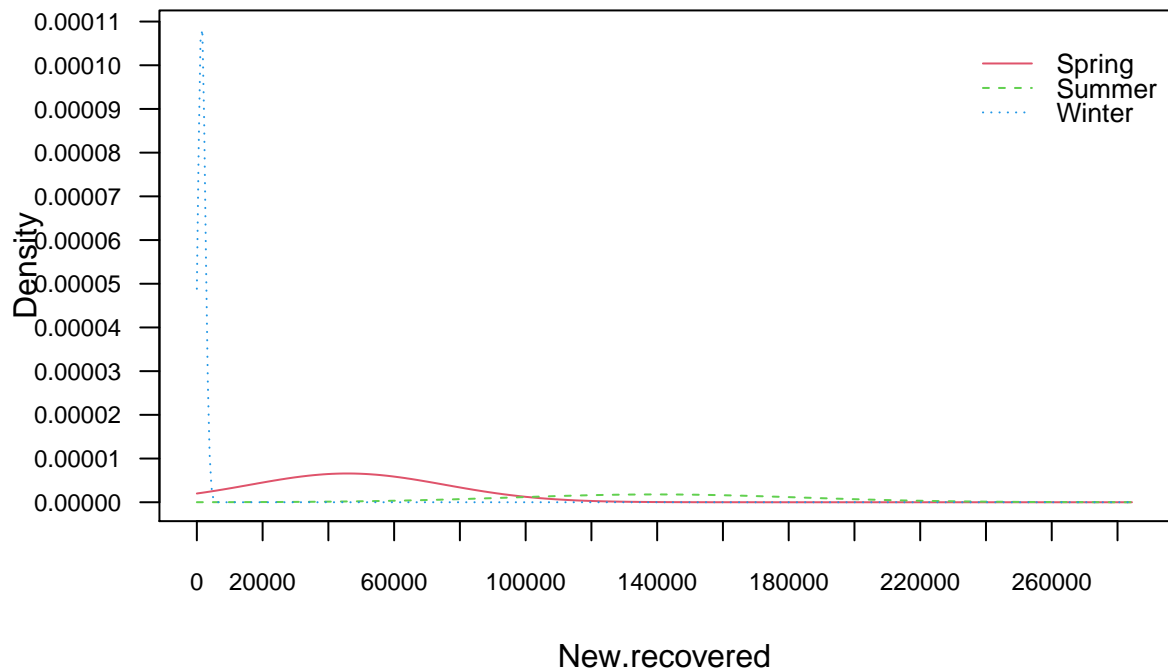
```
data <- read.delim('datos_nuevos_casos/regresion_train.csv', sep=";", head = TRUE)
```

E introducimos los datos, es importante factorizar la propiedad de “Season” para que funcione la función

```
nb <- naive_bayes(as.factor(Season) ~ New.cases + New.deaths + New.recovered, data)
plot(nb)
```







Ahora vamos a mostrar la tabla de probabilidades, para cada atributo se muestra el valor medio que tienen estos atributos para cada estación

```
nb$tables
```

```
##
## -----
##   ::: New.cases (Gaussian)
## -----
##
## New.cases      Spring      Summer      Winter
##      mean  92930.946 207459.216  4666.339
##      sd   27179.886 37531.138  6267.200
##
## -----
##   ::: New.deaths (Gaussian)
## -----
##
## New.deaths      Spring      Summer      Winter
##      mean  4932.5326 5103.0811  193.4237
##      sd   1527.4537 1273.4506  285.6735
##
## -----
##   ::: New.recovered (Gaussian)
## -----
##
## New.recovered      Spring      Summer      Winter
```

```
##          mean  45794.500 139715.784   1448.831
##          sd    29672.212  43986.753   1135.264
##
## -----
```

De esta forma, Naive bayes es capaz de calcular la probabilidad de que la muestra se trate de una estación u otra según el número de casos, muertes y recuperaciones diarias. Usando la función de predict nos devolverá directamente el resultado de la predicción.

Ahora realizaremos la validacion cruzada para ver la precisión de naive bayes

```
cvNaiveBayes <- lapply(folds, function(x){
  training_fold <- training_set[-x, ]
  test_fold <- training_set[x, ]
  clasificador <- naive_bayes(as.factor(Season) ~ New.cases + New.deaths + New.recovered, training_fold)
  y_pred <- predict(clasificador, newdata = test_fold)
  cm <- table(test_fold$Season, y_pred)
  precision <- (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])
  return(precision)
})
precisionNaiveBayes <- mean(as.numeric(cvNaiveBayes))
precisionNaiveBayes
```

```
## [1] 0.9345455
```

La precisión entonces es de 93.45%

Fuente: <https://rpubs.com/rdelgado/405322>

Clasificación mediante máquinas de vectores de soporte

Fuentes: <https://www.diegocalvo.es/svm-maquinas-de-vectores-de-soporte-en-r/>

Este método de clasificación se basa en la búsqueda un hiperplano que separe de forma óptima a todos los puntos de una clase, clasificandolos. El algoritmo de SVM tratará de buscar el hiperplano que tenga la máxima distancia posible con los puntos, de esta forma se podrá hacer una mejor clasificación de los datos, etiquetando cada dato dependiendo de en que lado del hiperplano se encuentra.

```
library(e1071)
# Ejecución del modelo SVM
data <- read.delim('datos_nuevos_casos/regresion_train.csv', sep=",", head = TRUE)

modelo = svm(as.factor(Season) ~ New.cases + New.deaths + New.recovered, data = data, kernel = "linear")
```

Hemos tratado de mostrar la gráfica pero no hemos sido capaces de hacerlo. Una vez sacado el modelo, podremos hacer las predicciones mediante la función predict, esta función devolverá directamente el valor de la predicción.

Ahora vamos a realizar la validacion cruzada para comprobar la eficacia de las máquinas de vectores de soporte

```
library(e1071)
cvKernelSVM <- lapply(folds, function(x){
  training_fold <- training_set[-x, ]
  test_fold <- training_set[x, ]
  clasificador <- svm(as.factor(Season) ~ ., data = training_fold, kernel = "linear", cost = 10)
  prediccion <- predict(clasificador, new=training_set)
  mc <- with(training_set, (table(prediccion, Season)))
  y_pred <- predict(clasificador, newdata = test_fold)
```

```

cm <- table(test_fold$Season, y_pred)
precision <- (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] + cm[1,2] + cm[2,1])
return(precision)
})

precisionKernelSVM <- mean(as.numeric(cvKernelSVM))
precisionKernelSVM

```

```
## [1] 0.9909091
```

Como podemos ver la precisión es de un 98.09%

Fuente: <https://rpubs.com/rdelgado/405322>

Conclusiones

Como hemos podido observar el árbol es mucho mas preciso que el resto de modelos. Los árboles de decisión además tienen la ventaja de que pueden ser usados tanto para predecir variables cualitativas y cuantitativas, además de que son simples de entender e interpretar. Sin embargo, los árboles de decisión pueden resultar algo inestables ante cualquier cambio en los datos de entrada, puede dar lugar a un árbol totalmente diferente.

Por otro lado, aunque las maquinas de soporte de vectores tenga una eficacia algo menor, de un 98%, son bastante eficaces igualmente. Además, las maquinas de soporte de vectores tienen la ventaja de que son bastante mas exactas en espacios de dimensiones altos. Aunque, tienen el inconveniente de que no son muy eficientes si el dataset es muy grande.

Por último, aunque el de Naive Bayes sea de los tres con un 93.45% el que menos tasa de acierto tenga, es de los clasificadores más útiles cuando presuponemos la independencia entre las variables en comparación con otros. Sin embargo, también puede ser considerada un problema si los aplicamos a los datos de la vida real, puesto que Naive Bayes asume la independencia de los predictores, y en la vida real es complicado obtener un conjunto de predictores que sean totalmente independientes.

Predicción no supervisada

Para estas predicciones hemos usado un dataset extraído de kaggle (https://www.kaggle.com/imdevskp/corona-virus-report?select=day_wise.csv) que muestra los datos del covid a nivel mundial divididos por días.

Las técnicas de predicción no supervisada tiene el objetivo de explorar un conjunto de datos para encontrar alguna estructura o forma de organizarlos. Por ello es muy frecuente emplear estas técnicas para agrupar datos con características o comportamientos similares.

En este apartado veremos algunas técnicas de aprendizaje no supervisado empleadas para la organización de los datos.

1. KMeans

Fuente: <https://rpubs.com/williamsurles/310847>, <https://www.geeksforgeeks.org/clustering-in-r-programming>

K-Means es una técnica iterativa de agrupamiento duro que utiliza un algoritmo de aprendizaje no supervisado. En él, el número total de grupos es predefinido por el usuario, y en base a la similitud de cada punto de datos, los puntos de datos se agrupan. Es decir, consiste en una agrupación de datos, en la que el conjunto de datos se divide en varios grupos llamados “clusters” en función de su similitud. Después de la segmentación de los datos se producen varios grupos de datos, y todos los objetos de un grupo comparten características comunes.

Cargamos librerías

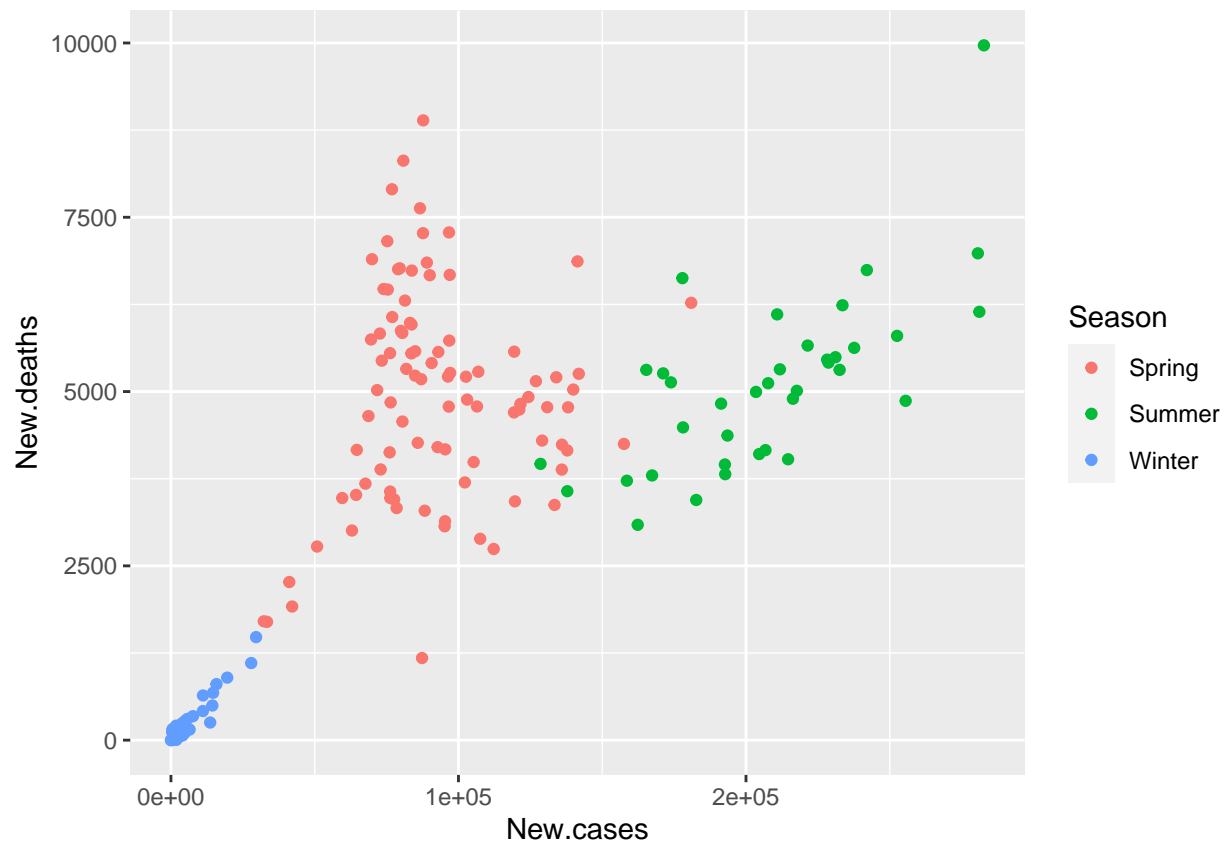
```
library(ggplot2)
```

Cargamos y visualizamos los datos

```
data <- read.delim('datos_nuevos_casos/regresion_train.csv', sep="," , head = TRUE)
head(data)
```

```
##   Date Confirmed Deaths Recovered Active New.cases New.deaths New.recovered
## 1    0         555      17        28   510         0         0         0
## 2    1         654      18        30   606        99         1         2
## 3    2         941      26        36   879       287         8         6
## 4    3        1434      42        39  1353       493        16         3
## 5    4        2118      56        52  2010       684        14        13
## 6    5        2927      82        61  2784       809        26         9
##   Deaths...100.Cases Recovered...100.Cases Deaths...100.Recovered
## 1                3.06                5.05                60.71
## 2                2.75                4.59                60.00
## 3                2.76                3.83                72.22
## 4                2.93                2.72               107.69
## 5                2.64                2.46               107.69
## 6                2.80                2.08               134.43
##   No..of.countries Season
## 1                6 Winter
## 2                8 Winter
## 3                9 Winter
## 4               11 Winter
## 5               13 Winter
## 6               16 Winter
```

```
ggplot(data, aes(New.cases, New.deaths, New.recovered, color = Season)) + geom_point()
```

Al interpretar los datos, podemos observar como hay 3 grupos de datos diferenciados. (1 = winter, 2 = Spring, 3 = Summer)

Realizamos una primera prueba de K-means con 2 cluster, para ello utilizamos todas las columnas del dataset exceptuando la de Seasons:

```
km.out <- kmeans(data[0:12], centers = 2, nstart = 20)
summary(km.out) # Inspect the result
```

```
##           Length Class  Mode
## cluster      188   -none- numeric
## centers        24   -none- numeric
## totss          1   -none- numeric
## withinss       2   -none- numeric
## tot.withinss   1   -none- numeric
## betweenss      1   -none- numeric
## size           2   -none- numeric
## iter           1   -none- numeric
## ifault         1   -none- numeric
```

```
print(km.out)
```

```
## K-means clustering with 2 clusters of sizes 57, 131
```

```
##
```

```
## Cluster means:
```

```
##   Date Confirmed   Deaths Recovered   Active New.cases New.deaths New.recovered
## 1  159  10691608 509100.4 5554008.6 4628499 180644.07   4919.825   120745.58
## 2   65   1672419 109665.5 548318.6 1014435  47360.61   2851.824   19737.11
```

```

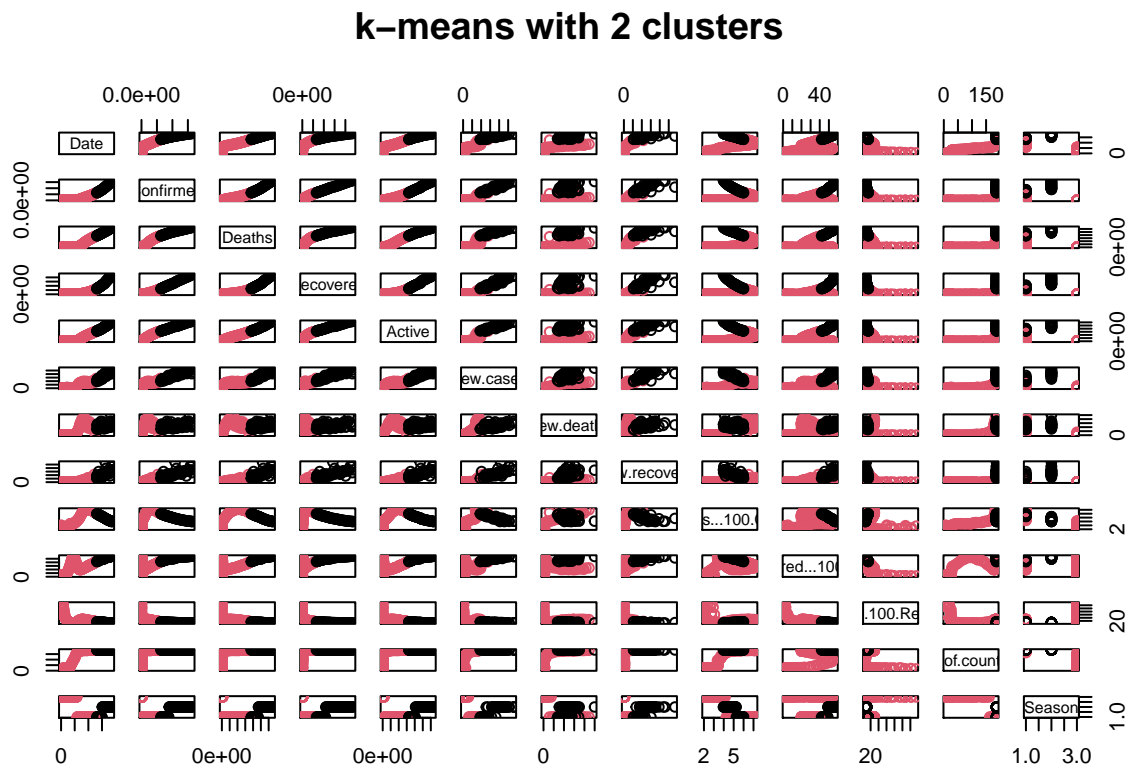
## Deaths...100.Cases Recovered...100.Cases Deaths...100.Recovered
## 1 4.933333 50.73877 9.907719
## 2 4.829008 27.21031 27.411527
## No..of.countries
## 1 187.0000
## 2 125.7939
##
## Clustering vector:
## [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 7.945756e+14 7.139665e+14
## (between_SS / total_SS = 75.9 %)
##
## Available components:
##
## [1] "cluster" "centers" "totss" "withinss" "tot.withinss"
## [6] "betweenss" "size" "iter" "ifault"

```

```

plot(data,
  col = km.out$cluster,
  main = "k-means with 2 clusters")

```

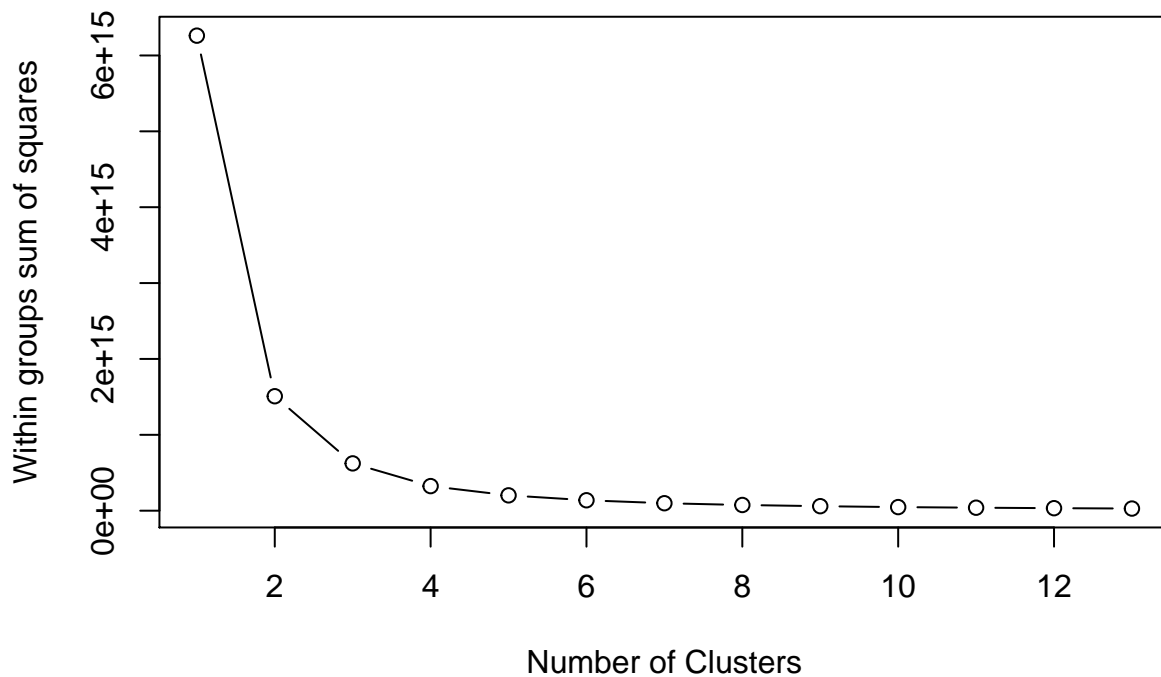


Como podemos observar en la consola R hay un buen ratio BSS/TSS *, ya que es del 75.9 %. Pero se podría mejorar.

(*) BSS -> la suma de las distancias al cuadrado de cada observación con la media de la muestra global, obtenemos total_SS. TSS -> la suma de las distancias cuadradas de estas tres medias a la media general, obtenemos between_SS.

Necesitamos elegir k, el número de clusters idóneo. Para ello hay un punto donde la curva SSE comienza a doblarse conocido como el punto del codo. Se cree que el valor x de este punto es un equilibrio razonable entre el error y el número de cúmulos. En nuestro caso, tras observar la gráfica podemos ver que es 3.

```
wss <- 0
for (i in 1:13) {
  km.out <- kmeans(x = data[0:12], centers = i, nstar=20)
  wss[i] <- km.out$tot.withinss
}
plot(1:13, wss, type = "b", xlab = "Number of Clusters", ylab = "Within groups sum of squares")
```

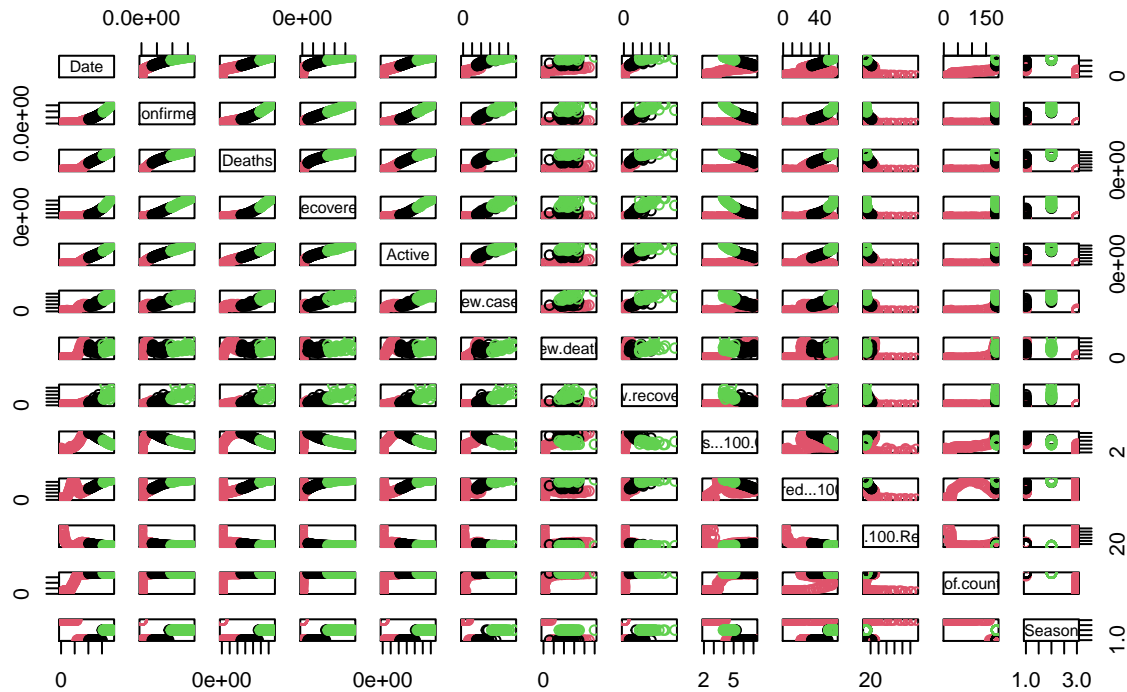


Como se puede ver en la gráfica empieza a decrementar en x=3, por lo que emplearemos k = 3.

```
k <- 3
km <- kmeans(data[0:12], centers = k, nstart = 20)
print(km)

## K-means clustering with 3 clusters of sizes 53, 102, 33
##
## Cluster means:
##   Date Confirmed   Deaths Recovered   Active New.cases New.deaths
## 1 128.0   6123787 366602.26   2596692 3160493.0 112750.60   4492.868
```


k-means clustering with 3 clusters



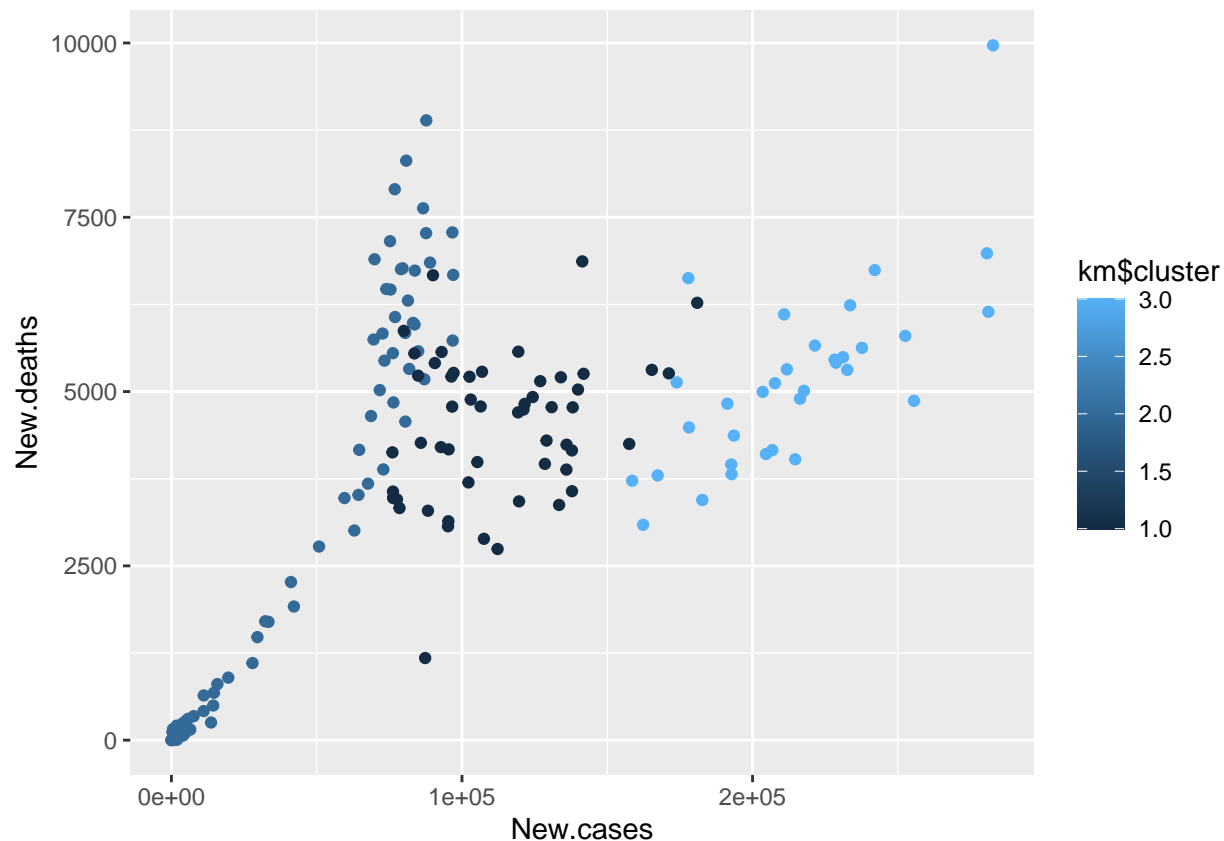
```
table(km$cluster, data$Season)
```

```
##
##      Spring Summer Winter
## 1      49         4       0
## 2      43         0      59
## 3       0        33       0
```

Al emplear $k = 3$ se ha obtenido un ratio BSS/TSS del 90%, el cual es mejor que el anterior ($k=2$).

Visualizamos el resultado:

```
ggplot(data, aes(New.cases, New.deaths, New.recovered, color = km$cluster)) + geom_point()
```



Como se puede ver en la gráfica han salido 3 grupos diferenciados de datos, los cuales se corresponden con la división inicial por Seasons. De esto podemos concluir que la agrupación de k-means ha tenido un resultado satisfactorio, ya que equivale a las 3 Seasons.

2. Principal component analysis

Fuente: https://rpubs.com/Cristina_Gil/PCA, <https://lgatto.github.io/IntroMachineLearningWithR/unsupervised-learning.html#principal-component-analysis-pca>

PCA es una de las técnicas de aprendizaje no supervisado, las cuales suelen aplicarse como parte del análisis exploratorio de los datos.

Nosotros emplearemos esta técnica como herramienta para la visualización de datos, aunque también se suele emplear para la reducción de dimensiones (variables).

En este caso cabe destacar que, aunque hemos investigado y estudiado como sería el proceso, debido a un fallo persistente con la instalación de la librería “FactoMineR” no nos ha sido posible ejecutar la función PCA. Por tanto, nos resulta imposible visualizar la técnica. Aún así, hemos querido dejar constancia del trabajo realizado:

Procedemos a realizar las instalaciones pertinentes:

```
library("FactoMineR")
```

Cargamos los datos:

```
data <- read.delim('datos_nuevos_casos/regresion_train_kmeans.csv', sep=",", head = TRUE)
head(data)
```

```
##   Date Confirmed Deaths Recovered Active New.cases New.deaths New.recovered
```

```
## 1 0 555 17 28 510 0 0 0
## 2 1 654 18 30 606 99 1 2
## 3 2 941 26 36 879 287 8 6
## 4 3 1434 42 39 1353 493 16 3
## 5 4 2118 56 52 2010 684 14 13
## 6 5 2927 82 61 2784 809 26 9
## Deaths...100.Cases Recovered...100.Cases Deaths...100.Recovered
## 1 3.06 5.05 60.71
## 2 2.75 4.59 60.00
## 3 2.76 3.83 72.22
## 4 2.93 2.72 107.69
## 5 2.64 2.46 107.69
## 6 2.80 2.08 134.43
## No..of.countries New.Cases.gt.5000 New.Deaths.gt.5000 New.Recover.gt.5000
## 1 6 0 0 0
## 2 8 0 0 0
## 3 9 0 0 0
## 4 11 0 0 0
## 5 13 0 0 0
## 6 16 0 0 0
## Season
## 1 1
## 2 1
## 3 1
## 4 1
## 5 1
## 6 1
```

Antes de aplicar un PCA, las observaciones tienen que moverse al centro del eje de coordenadas, esto es, centrarlas para que tengan media 0, para así eliminar posibles bias en las mediciones. La función `prcomp()` es una de las múltiples funciones en R que realizan PCA, `prcomp()` centra las variables para que tengan media 0

```
pca <- prcomp(data)
```

Otro de los outputs de la función `prcomp()` es la desviación estándar de cada componente principal:

```
pca$sdev
```

```
## [1] 5.765574e+06 4.877041e+05 2.259462e+04 1.827776e+04 9.112740e+03
## [6] 8.191240e+02 3.335632e+01 1.633777e+01 5.735689e+00 1.940097e+00
## [11] 2.627300e-01 2.515161e-01 1.946349e-01 1.415441e-01 9.793089e-02
## [16] 4.169821e-10
```

La varianza explicada por cada componente principal la obtenemos elevando al cuadrado la desviación estándar:

```
pca$sdev^2
```

```
## [1] 3.324185e+13 2.378553e+11 5.105168e+08 3.340765e+08 8.304203e+07
## [6] 6.709642e+05 1.112644e+03 2.669226e+02 3.289812e+01 3.763976e+00
## [11] 6.902707e-02 6.326035e-02 3.788274e-02 2.003473e-02 9.590460e-03
## [16] 1.738741e-19
```

Aplicamos PCA

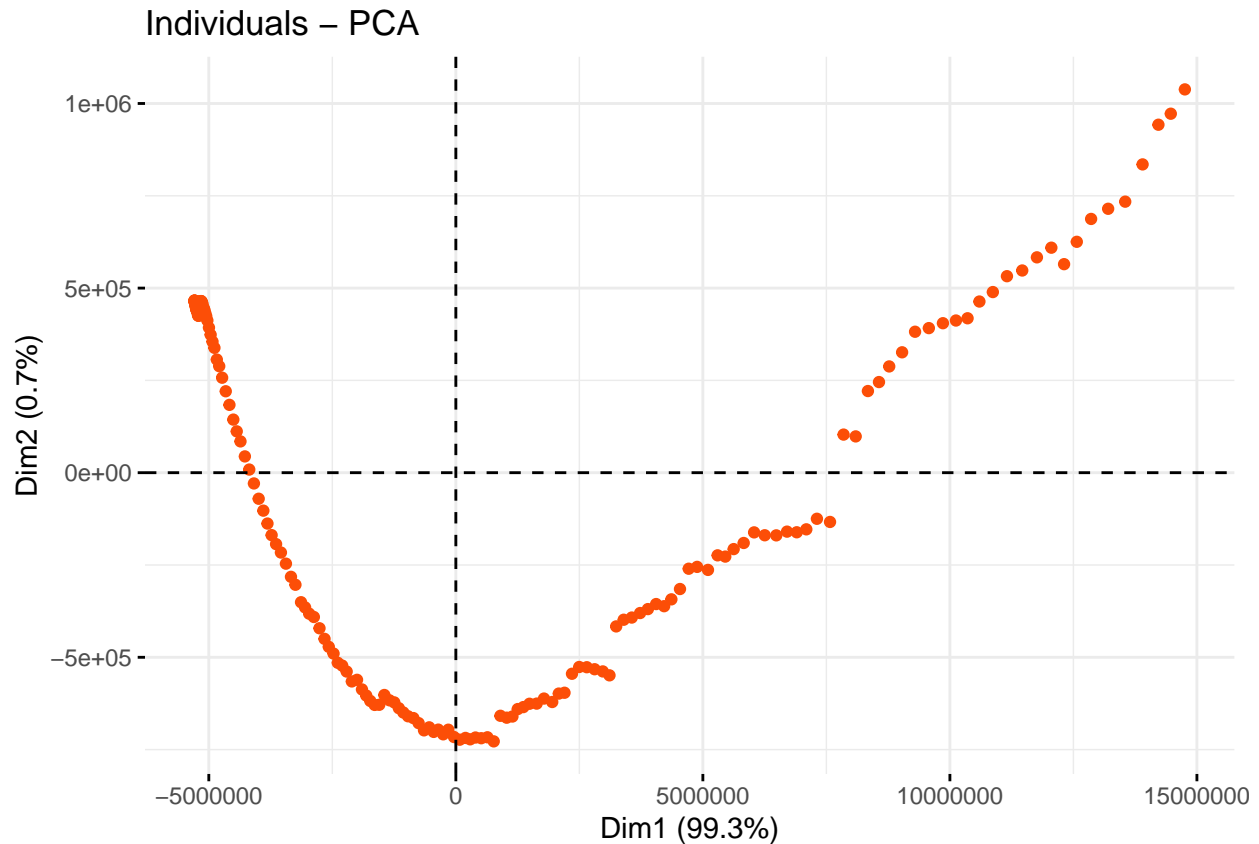
```
pca2.nci <- PCA(X = data, scale.unit = TRUE, ncp = 64, graph = FALSE)
```

Mostramos dos ejemplos para representar las observaciones sobre las dos primeras componentes principales:

```
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
fviz_pca_ind(pca, geom.ind = "point",  
             col.ind = "#FC4E07",  
             axes = c(1, 2),  
             pointsize = 1.5)
```



DBSCAN

Fuente: https://rpubs.com/elias_jurgen/605966

Es un método de clusterización adecuado para buscar patrones de agrupación en el espacio físico. Este algoritmo agrupa los puntos que están más cercanos respecto a la distancia euclidiana, además para este algoritmo se tiene que cada cluster contendrá un mínimo de puntos.

Este método necesita sólo dos parámetros

- `eps`: Esta es la distancia que se tomará como radio de los clusters. Este parámetro se puede elegir “correctamente” basándose en la distancia del dataset (utilizando un K-distance Plot). Es preferible utilizar valores pequeños

- `minPoints`: Mínimo de números que deben estar en un grupo para que el algoritmo lo tome como un cluster. Este número se puede elegir basándose en el número de dimensiones del dataset.

- `D` es el número de dimensiones del dataset. Este parámetro debe ser proporcional al tamaño del dataset pero nunca es menor que 3

Procedemos a realizar las instalaciones pertinentes:


```
library("dbscan")
```

Cargamos los datos. Sólo hay 3 tipos de Seasons, así que lo ideal sería obtener esos 3 clusters. Tenemos 16 variables, una de ellas es "Seasons" que es la clasificación a la que trataríamos de llegar, por ello la vamos a eliminarla del dataset. Una vez eliminada tendremos 15 variables restantes en la base, así que tomaremos 16 como el mínimo número de puntos.

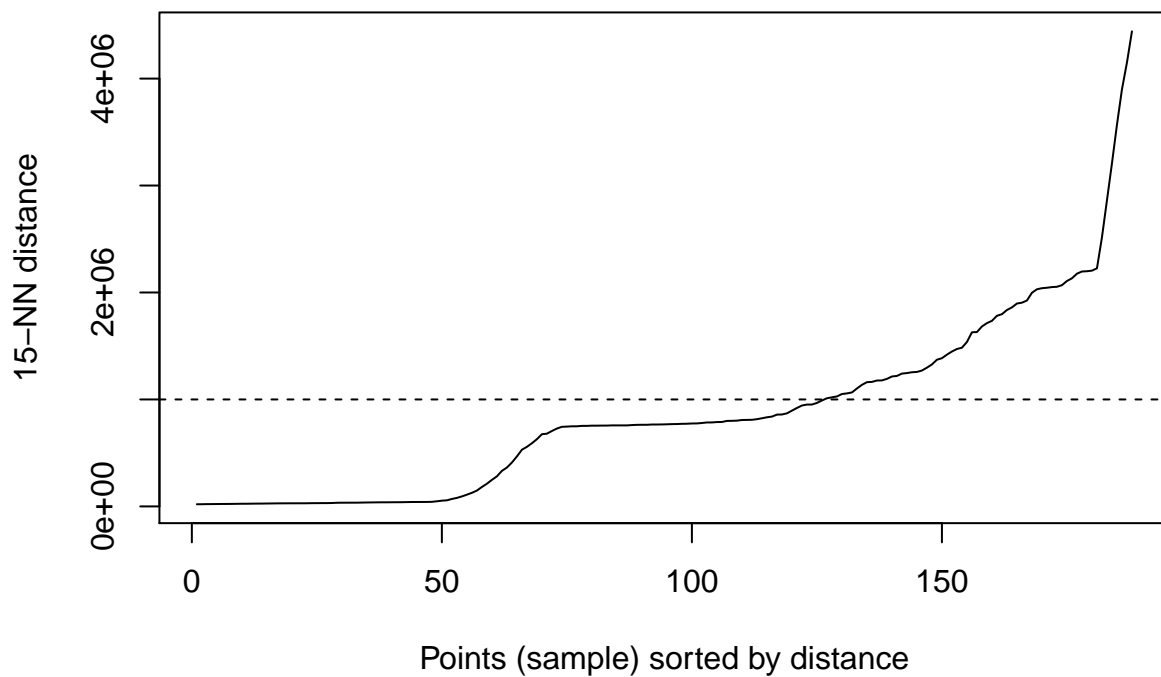
```
data <- read.delim('datos_nuevos_casos/regresion_train.csv', sep=",", head = TRUE)
data$Season <- NULL
head(data)
```

```
##   Date Confirmed Deaths Recovered Active New.cases New.deaths New.recovered
## 1    0         555     17        28    510         0         0         0
## 2    1         654     18        30    606         99         1         2
## 3    2         941     26        36    879        287         8         6
## 4    3        1434     42        39   1353        493        16         3
## 5    4        2118     56        52   2010        684        14        13
## 6    5        2927     82        61   2784        809        26         9
##   Deaths...100.Cases Recovered...100.Cases Deaths...100.Recovered
## 1                3.06                5.05                60.71
## 2                2.75                4.59                60.00
## 3                2.76                3.83                72.22
## 4                2.93                2.72               107.69
## 5                2.64                2.46               107.69
## 6                2.80                2.08               134.43
##   No..of.countries
## 1                 6
## 2                 8
## 3                 9
## 4                11
## 5                13
## 6                16
```

Ahora, para elegir el radio de los grupos utilizaremos un K-Distance Plot.

```
#df: dataset sin variable "species"
#k: el número mínimo de puntos que elegimos

kNNdistplot(data, k = 15)
abline(h = 1000000, lty = 2)
```



eps=1000000

Por lo que se observa en la gráfica tomaremos `eps=1000000`

Al aplicar la función y representar gráficamente el resultado se puede observar 1 clusters. Se ve claramente como hemos obtenido un número erróneo de clusters, esto puede deberse al valor de `eps`.

```
cl<-dbscan(data,eps=1000000,MinPts = 5)
```

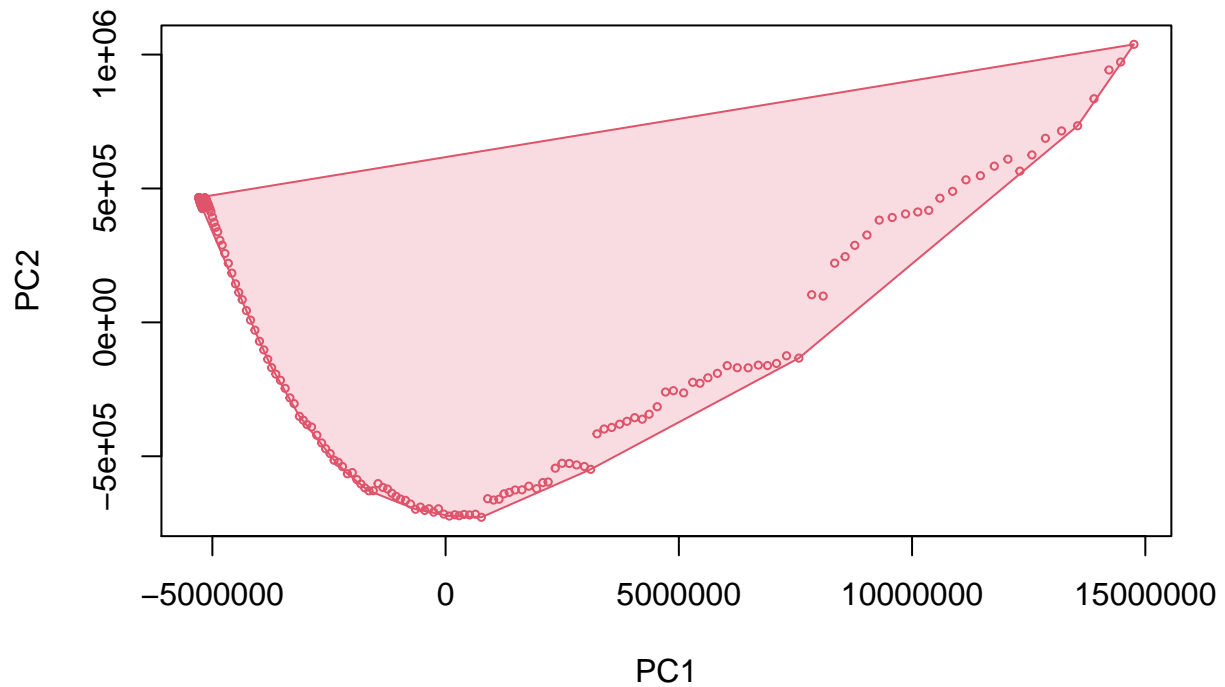
```
## Warning in dbscan(data, eps = 1e+06, MinPts = 5): converting argument MinPts
## (fpc) to minPts (dbscan)!
```

```
unique(cl$cluster)
```

```
## [1] 1
```

```
hullplot(data,cl$cluster, main = "Convex cluster Hulls, eps=1000000")
```

Convex cluster Hulls, eps=1000000



eps=3000000

Tomaremos eps=3000000

Se procede a aplicar la función ahora con un número más elevado de eps, pero el resultado sigue persistiendo.

```
cl<-dbscan(data,eps=3000000,MinPts = 5)
```

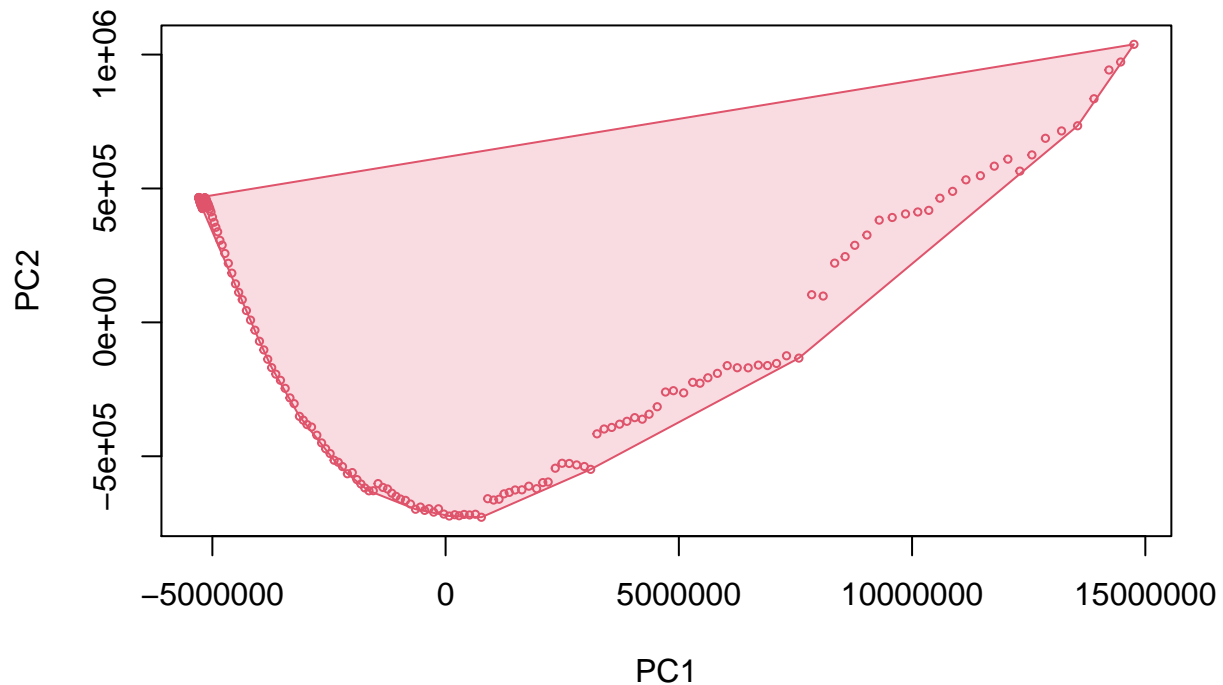
```
## Warning in dbscan(data, eps = 3e+06, MinPts = 5): converting argument MinPts  
## (fpc) to minPts (dbscan)!
```

```
unique(cl$cluster)
```

```
## [1] 1
```

```
hullplot(data,cl$cluster, main = "Convex cluster Hulls, eps=3000000")
```

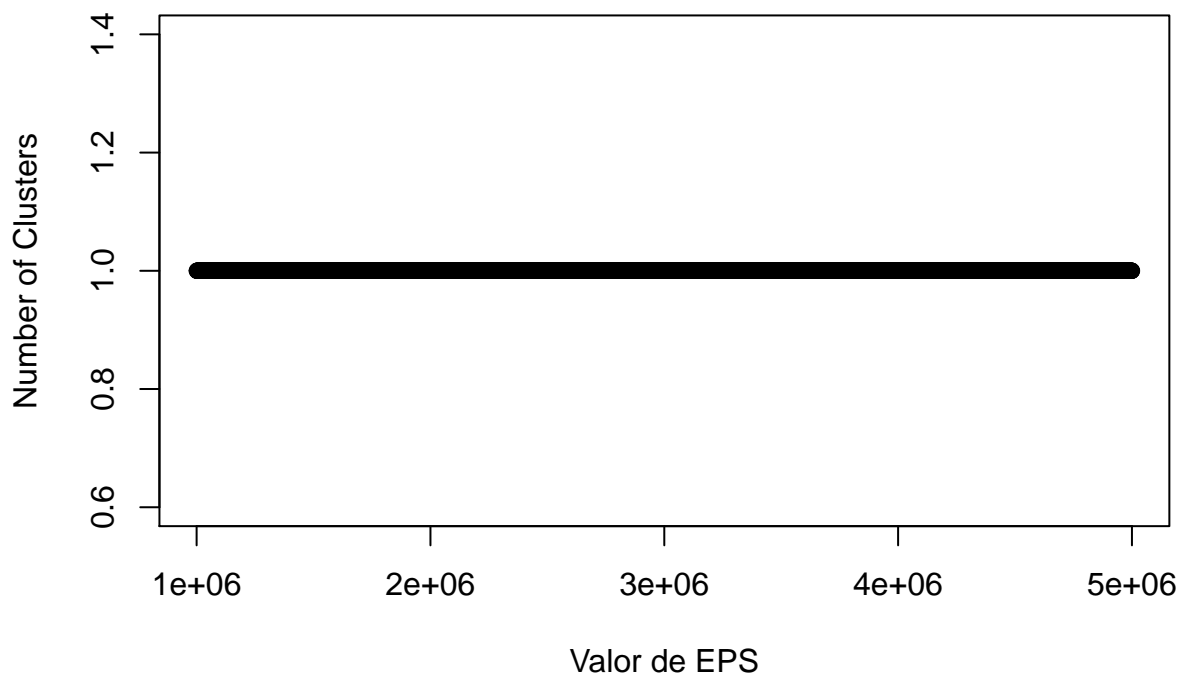
Convex cluster Hulls, eps=3000000



Podemos observar como siempre obtenemos 1 cluster independientemente del valor del eps

Por último, hacemos la prueba con muchos valores de eps y mostramos el número de clusters devuelto. Los valores de eps van desde 1000000 hasta 5000000, lo sabemos gracias a la gráfica anterior generada mediante kNNdistplot, pero de 1000 en 1000 para agilizar el proceso. Podemos observar como siempre devuelve 1 cluster.

```
wss <- 0
value <- 0
i <- 1000000
index <- 0
while(i<=5000000) {
  km.out <- dbSCAN(data,eps=i,MinPts = 5)
  wss[index] <- km.out$cluster
  value[index] <- i
  i<-i+1000
  index <- index + 1
}
plot(value, wss, type = "b", xlab = "Valor de EPS", ylab="Number of Clusters")
```



Como se puede observar, para los diferentes valores del rango de eps que se ofrece el número de clusters persite de forma continua en 1.

Heatmaps

Fuente: https://rpubs.com/Joaquin_AR/310338

Los heatmaps son el resultado obtenido al representar una matriz de valores en la que, en lugar de números, se muestra un gradiente de color proporcional al valor de cada variable en cada posición. Se consigue representar más información que con un simple dendrograma y se facilita la identificación visual de posibles patrones característicos de cada cluster.

Procedemos a realizar las instalaciones pertinentes:

```
library(ComplexHeatmap)
```

```
## Loading required package: grid
## =====
## ComplexHeatmap version 2.6.2
## Bioconductor page: http://bioconductor.org/packages/ComplexHeatmap/
## Github page: https://github.com/jokergoo/ComplexHeatmap
## Documentation: http://jokergoo.github.io/ComplexHeatmap-reference
##
## If you use it in published research, please cite:
## Gu, Z. Complex heatmaps reveal patterns and correlations in multidimensional
## genomic data. Bioinformatics 2016.
##
## This message can be suppressed by:
```

```
## suppressPackageStartupMessages(library(ComplexHeatmap))
## =====

library(viridis)

## Loading required package: viridisLite
colores <- magma(256)

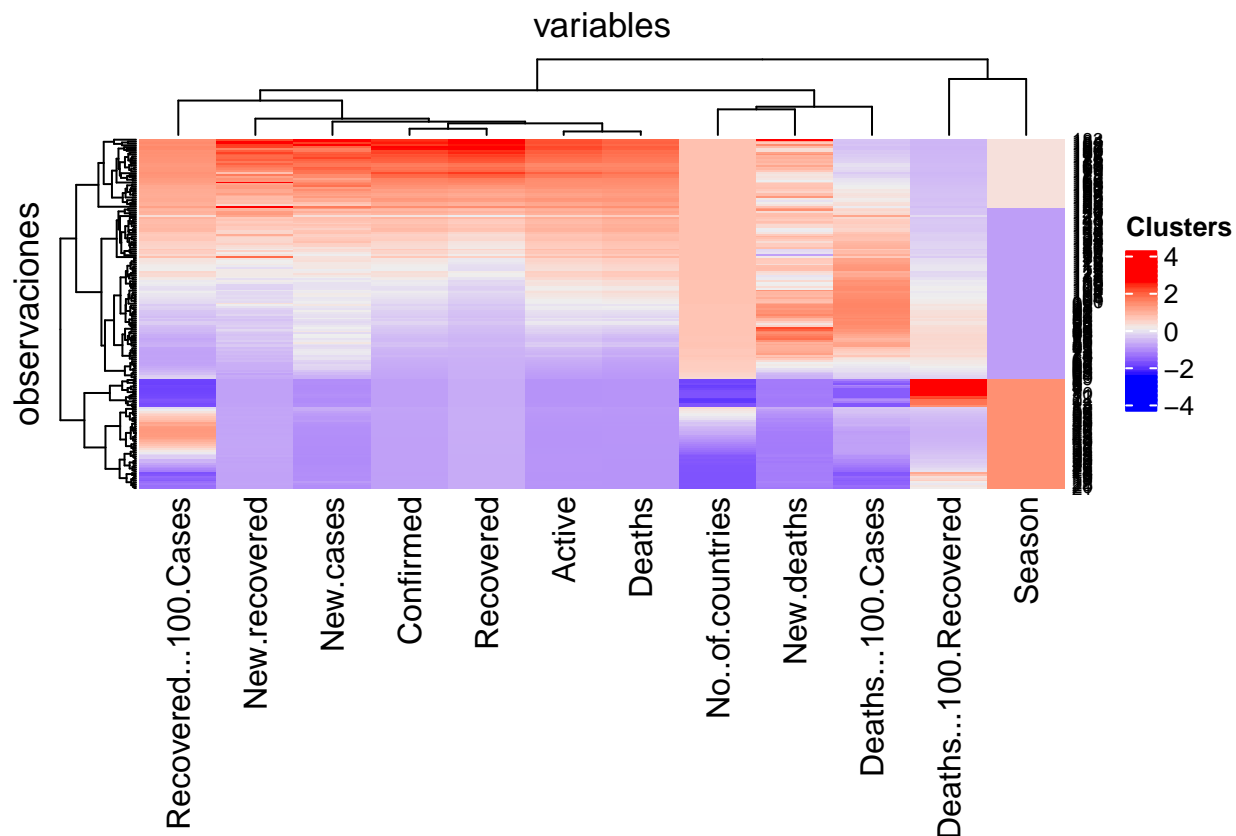
Cargamos los datos
data <- read.delim('datos_nuevos_casos/regresion_train.csv', sep=",", head = TRUE, row.names = 1,
                  as.is=TRUE)
data$Season <- as.numeric(as.factor(data$Season))
head(data)

##   Confirmed Deaths Recovered Active New.cases New.deaths New.recovered
## 0         555      17        28    510         0         0         0
## 1         654      18        30    606        99         1         2
## 2         941      26        36    879       287         8         6
## 3        1434      42        39   1353       493        16         3
## 4        2118      56        52   2010       684        14        13
## 5        2927      82        61   2784       809        26         9
## Deaths...100.Cases Recovered...100.Cases Deaths...100.Recovered
## 0              3.06              5.05              60.71
## 1              2.75              4.59              60.00
## 2              2.76              3.83              72.22
## 3              2.93              2.72             107.69
## 4              2.64              2.46             107.69
## 5              2.80              2.08             134.43
## No..of.countries Season
## 0              6      3
## 1              8      3
## 2              9      3
## 3             11      3
## 4             13      3
## 5             16      3

data <- as.matrix(data)
data <- scale(data)
```

Podemos observar como la columna Seasons tiene 3 cluster a diferencia de las demas columnas.

```
Heatmap(matrix = data, name = "Clusters",
        row_title = "observaciones",
        column_title = "variables",
        row_names_gp = gpar(fontsize = 7),
        clustering_distance_columns = "euclidean",
        clustering_distance_rows = "euclidean",
        clustering_method_columns = "average",
        clustering_method_rows = "average")
```



Conclusiones

Como hemos podido observar de manera general al emplear las diferentes técnicas de aprendizaje no supervisado, este conjunto de datos se puede organizar en 3 grupos diferenciados.

Al emplear K-means se pudo ver gráficamente como al pintar los datos en función de la columna Seasons en un inicio, surgían 3 grupos separados. Después al eliminar del conjunto de datos dicha columna y aplicar el algoritmo de k-means se han obtenido 3 clusters que se identifican con los 3 tipos de la columna Seasons. Por lo que se puede concluir que el algoritmo ha realizado un buen trabajo de agrupación.

En el caso de DBSCAN nos encontramos con la problemática técnica anteriormente descrita (fallo de instalación de librería), por la cual no pudimos ver resultados.

Aplicando PCA, podemos concluir que no es un método adecuado para clasificar nuestro conjunto de datos. Esto creemos que es debido a que tienen demasiado ruido y no hemos podido encontrar una manera efectiva de reducirlo y afinar el resultado.

Por último, al aplicar Heatmaps, hemos podido confirmar la agrupación de esos 3 grupos que se mostraron con k-means.

En general, hemos obtenido resultados aceptables y dentro de lo que esperábamos.

BIGML

Ahora vamos a tratar una herramienta en la nube llamada BIGML, en concreto, nos centraremos en la capacidad de predicción de esta aplicación. Usaremos para ello nuestro dataset, pero para que podamos hacer el estudio sobre las variables que hemos usado en el aprendizaje supervisado, tendremos que generar un nuevo dataset dentro de la aplicación.

DATASET CONFIGURATION

Dataset name

Size : 15.33 KB

regresion_train v1

100%

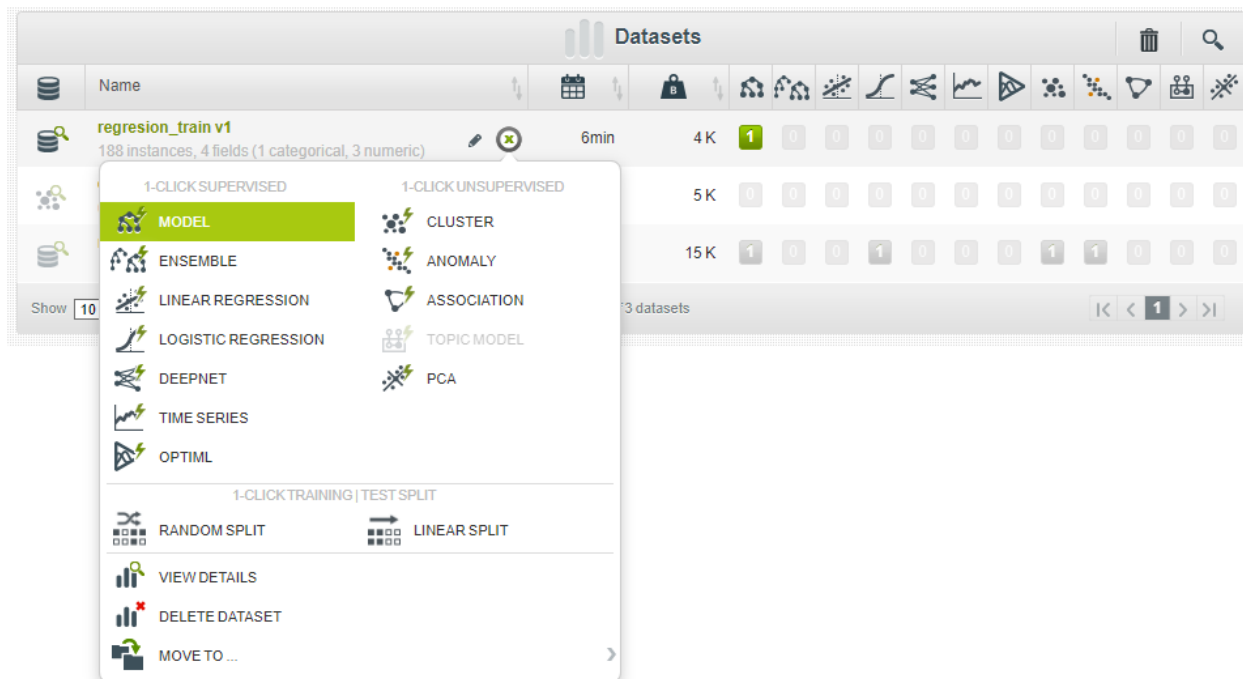
Create dataset

Name	Type	Instance 1	Instance 2	Instance 3
Date	1 2 3	0	1	2
Confirmed	1 2 3	555	654	941
Deaths	1 2 3	17	18	26
Recovered	1 2 3	28	30	36
Active	1 2 3	510	606	879
New cases	1 2 3	0	99	287
New deaths	1 2 3	0	1	8
New recovered	1 2 3	0	2	6
Deaths / 100 Cases	1 2 3	3.06	2.75	2.76
Recovered / 100 Cases	1 2 3	5.05	4.59	3.83

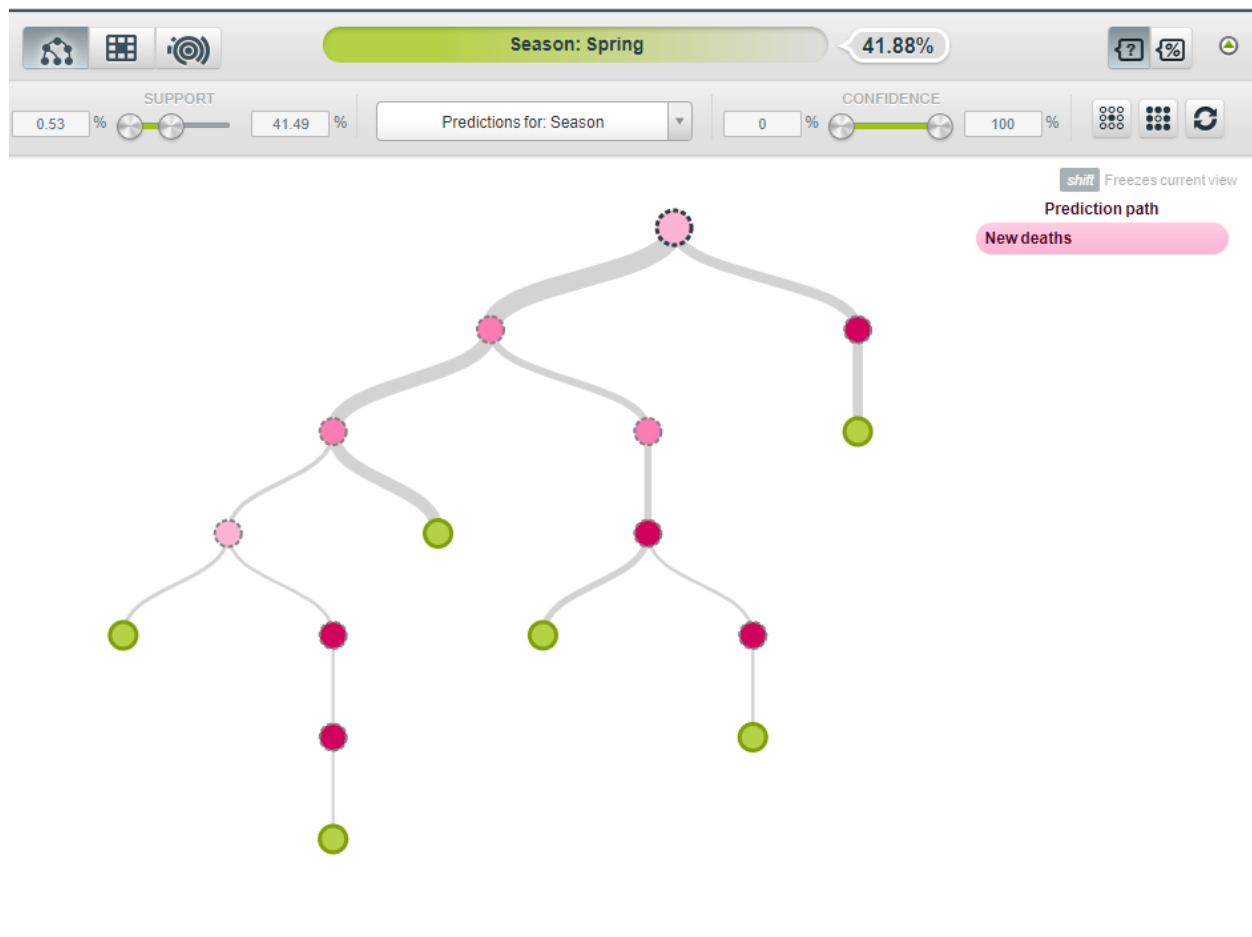
Show 10 fields1 to 10 of 16 fields

Figure 1: Deselección de atributos que no queremos

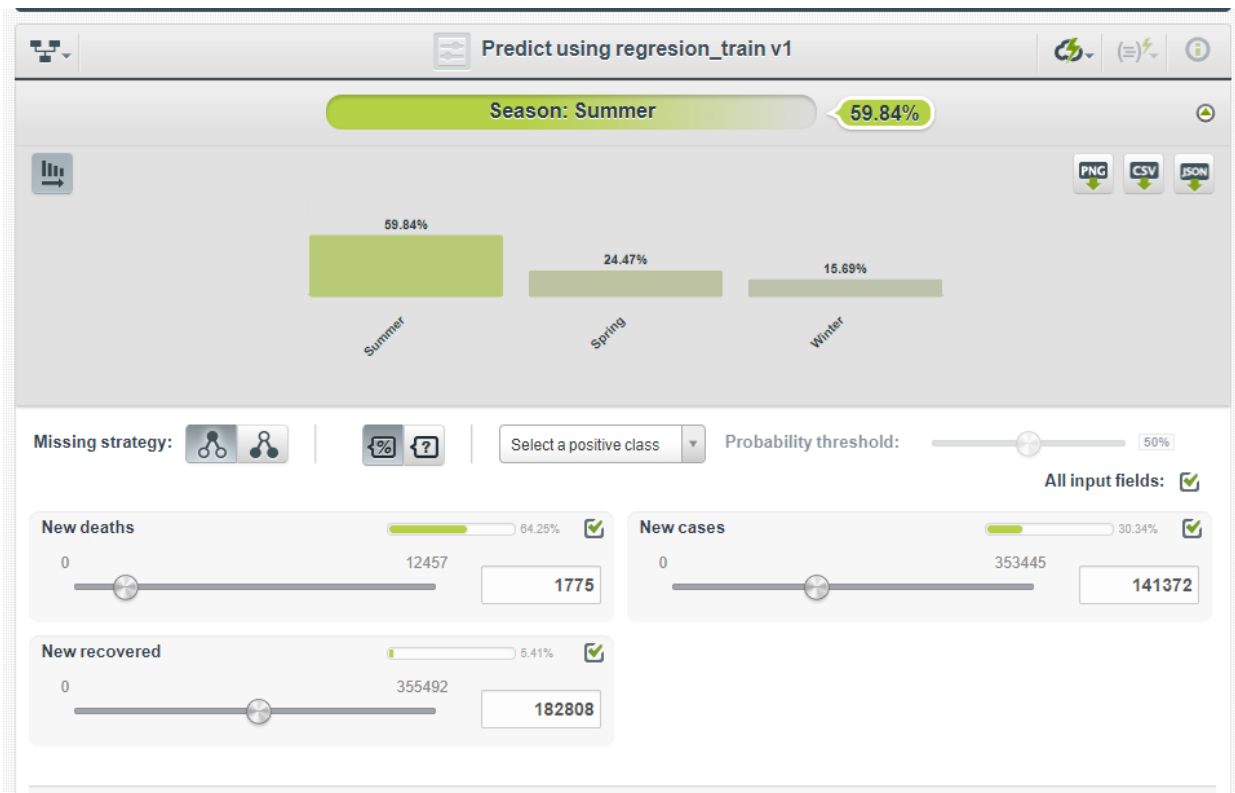
Como es lógico, tendremos que crear un modelo para poder hacer la predicción, por lo que mediante la interfaz crearemos dicho modelo



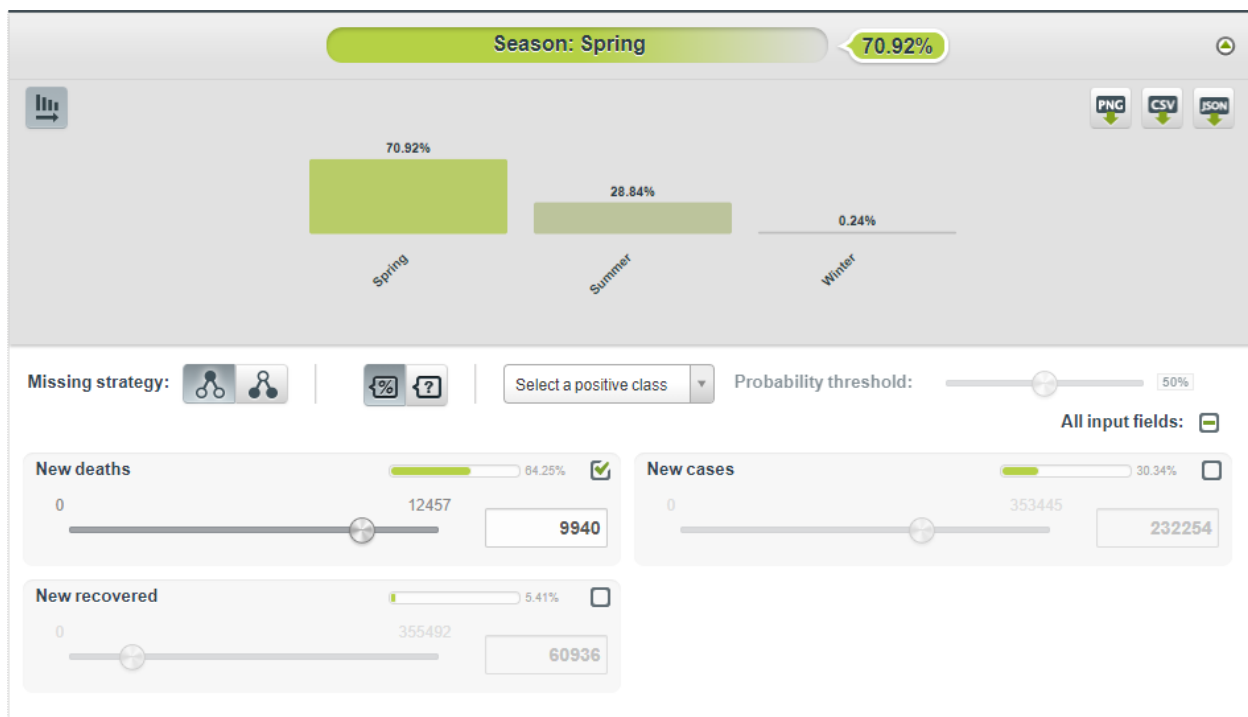
Como podemos ver a continuación el modelo que ha generado es un árbol

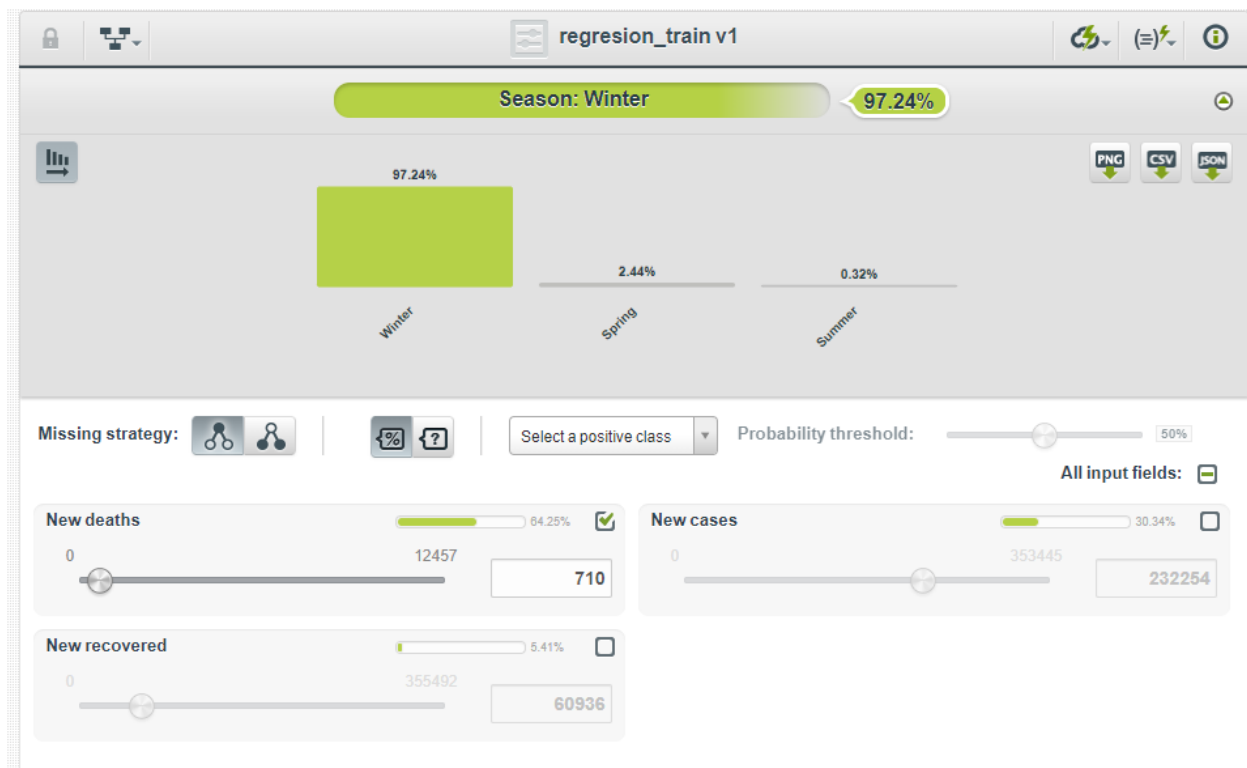


Podemos hacer las predicciones de varias formas distintas, para empezar la mas simple es una interfaz que mediante un pequeño formulario podemos introducir los valores de los datos y arriba nos mostrará una pequeña gráfica indicando la probabilidad de que pertenezca a una categoría u otra.



Incluso como podemos ver, tenemos la opción de desactivar los atributos que queramos, si por ejemplo desactivamos los atributos de “new cases” y de “new recovered”, observamos que hay 2 franjas bien definidas en donde se define claramente a que clase pertenece. La primera es en invierno donde no había una gran cantidad de tasa de muertos y la otra es en primavera, donde había una gran cantidad de muertes diarias, sin embargo en verano esa tasa varía bastante, por lo que aparece, aunque sea menos probable, en estas dos franjas.





Después, podemos realizar una predicción mediante preguntas, es decir, que la interfaz irá preguntando cada parámetro uno a uno. Al fin y al cabo la finalidad es la misma, predecir un caso concreto.

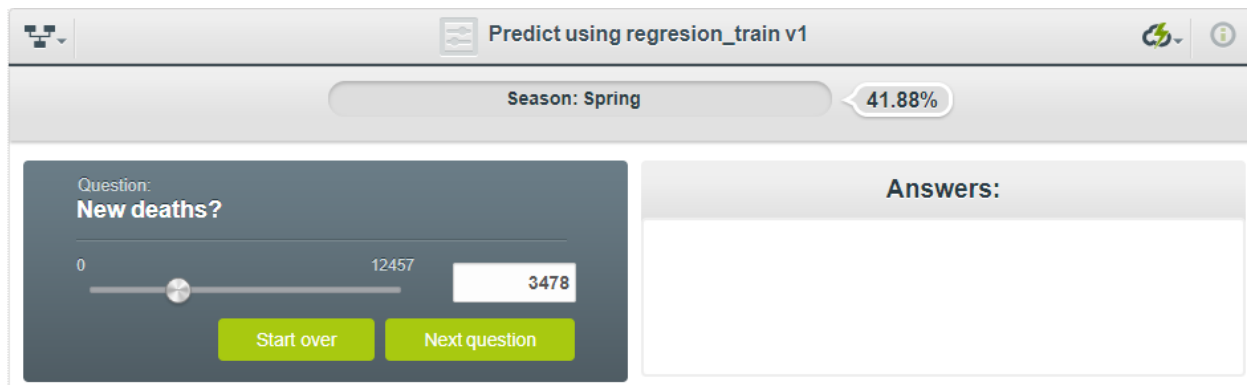







Figure 2: Predicción por preguntas

Lo mas interesante, es la opción de “Batch predict”, que nos permite introducir un dataset para que prediga el valor que estamos analizando de cada uno de los datos. Ahorrando sin duda bastante tiempo.



Regression_train V1




Regression_train


Configuration


Output preview

Date	Confirmed	Deaths	Recovered	Active	New cases	New deaths	New recovered	Deaths / 100 Cases	Recovered / 100 Cases	Deaths / 100 Rec
0	555	17	28	510	0	0	0	306	505	
1	654	18	30	606	99	1	2	275	459	
2	941	26	36	879	287	8	6	276	383	
3	1434	42	39	1353	493	16	3	293	272	
4	2118	56	52	2010	684	14	13	264	246	


Download batch prediction


Output dataset

Esta opción devolverá un dataset nuevo con una columna nueva con la predicción de cada dato.