# Proyecto de elaboración de un Dashboard en Shiny para el análisis de AirBnb

## Introducción

El análisis de datos es un campo crucial en la ciencia de datos y la toma de decisiones basada en datos en muchas áreas tanto académicas como comerciales. Tanto R como Python son lenguajes de programación poderosos y ampliamente utilizados en el análisis de datos. Sin embargo, aunque ambas herramientas sean muy buenas en el campo de análisis de datos por sí mismas de datos a nivel general, muchas veces necesitamos complementarlo con paquetes o librerías externas, incluso a veces haciendo uso de otros lenguajes de programación para procesos en los cuales pueden funcionar mejor que Python o R. En este trabajo haremos uso de tanto librerías externas como del lenguaje C++ para realizar un procedimiento de análisis de datos.

## Objetivos del proyecto

Con este proyecto queremos crear una herramienta orientada tanto para propietarios de alojamientos con licencia turística como para personas con intención de inversión de en ese sector que sirva para estudiar la situación del mercado, la competencia o posibles oportunidades de negocio, dentro de la ciudad de Barcelona. Esta herramienta será un dashboard que permita filtrar según las características que se quieran estudiar y permita visualizar la situación geográfica de los alojamientos con esos filtros.

### Necesidades del proceso

Para la creación de la herramienta debemos tener en cuenta diversos factores:

**Definir requisitos y funcionalidades**

Proceso de identificar y documentar las necesidades específicas y características que debe cumplir el proyecto o sistema. Es el proceso que estamos realizando en esta parte del análisis del programa.

**Buscar una fuente de datos relevante**

Encontrar y acceder a conjuntos de datos que proporcionen la información necesaria para un análisis o proyecto. Debemos evaluar elementos como la calidad, relevancia, tamaño, etc.

**Limpiar los datos**

Proceso de preparar los datos para el análisis eliminando o corrigiendo los datos erróneos, incompletos, duplicados o irrelevantes. En nuestro caso es fundamental, ya que nuestro proyecto implica preparar una plataforma capaz de filtrar de forma continua.

**Elaborar un interfaz**

Diseño o desarrollo de la interfaz para la aplicación. Es el eje central del proyecto.

**Optimizar el código para rendimiento y escalabilidad**

Modificaciones y ajustes para que el código funcione de forma ágil.

En caso de que esto fuera una propuesta real para el lanzamiento de la herramienta deberíamos tener en cuenta también otros elementos como:

- Seguridad y privacidad de datos

- Pruebas y validación

- Elaborar documentación completa

## Datos utilizados

Los datos que se van a utilizar provienen de Inside Airbnb, una organización sin animo de lucro, impulsada con la misión de informara las comunidades sobre el impacto y la realidad del alojamiento a corto plazo. Estos han sido extraidos de la propia plataforma de Airbnb con funciones de scrapping. Se omite la información privada. Los datos tienen licencia de “fair use”.El análisis se centra en un conjunto de datos que refleja los listados de alojamiento en activo extraídos de la plataforma Airbnb, específicamente para la ciudad de Barcelona. Este conjunto incluye registros detallados que comprenden información geográfica, precios, características del alojamiento y amenidades ofrecidas. Las variables clave que serán examinadas incluyen la latitud y longitud para la ubicación espacial y el precio por noche como indicador económico.

## Herramientas a utilizar para el proyecto

En el proyecto vamos a utilizar las siguientes herramientas:

**R**

Como hemos dicho anteriormente, en procesos de análisis de datos los lenguajes de programación con mas soporte son tanto R como Python. Python es un lenguaje mas universal y tiene mas fines, sin embargo en este proyecto vamos a priorizar tanto la manipulación de datos como la visualización por lo que R y el paquete de tidyverse, puede ser mejor herramienta en este caso.

**Shiny**

La herramienta final que queremos crear es un dashboard interactivo. Podríamos optar por un programa externo, pero es mas conveniente hacer uso de algún paquete en R. Shiny es uno de los paquetes con mas recursos y funciona a través de una aplicación web interactiva, lo que nos permite crear nuestro objetivo.

**C**

Para ciertos calculos e interacciones que requieran procesamientos de alto rendimiento puede ser mas conveniente utilizar C++. Haremos alguna prueba y compararemos los tiempos de procesado de R vs C++.

## Resultados esperados

Los resultados finales de nuestro proceso deben contener:

**Opciones de filtrado de todas las variables y con todos los valores**

Queremos que la herramienta tenga flexibilidad total y que permita cualquier filtro y con cualquier variable.

**Visualización con mapa**

Para poder ver la localización geográfica de las diferentes entradas además de el

**Visualización con tablas**

La herramienta debe permitir poder ver las entradas al completo para ofrecer la capacidad de encontrar patrones e información conveniente.

**Usabilidad**

La herramienta tiene que ser usable y tener un buen diseño.

## Análisis KDE

El principal valor que aporta nuestro proyecto es que se basa en la estimación de densidad kernel como primer factor visual a la hora de analizar el dashboard.

Esta es una técnica utilizada en estadística y análisis de datos para estimar la función de densidad de probabilidad de una variable aleatoria continua. En programación y análisis de datos, se aplica comúnmente para suavizar datasets, especialmente en la visualización de datos, donde se busca entender la distribución subyacente de los datos.

En el contexto de programación, KDE se implementa mediante algoritmos que toman un conjunto de datos y, utilizando una función kernel (en nuestro caso la Gaussiana) generan una estimación de la desnsidad. Es muy útil cuando se quiere obtener una visión general de la distribución y el agrupamiento de los puntos.

En nuestro caso concretamente hacemos un análisis KDE de coordenadas, el cual toma dos variables, la longitud y la latitud para visualizarse.

En cuanto a su fórmula matemática es la siguiente:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$$

## Limpieza de datos

En cuanto a la preparación de los datos, se llevará a cabo un meticuloso proceso de limpieza para garantizar la integridad y la calidad de la información analizada. Este proceso abarca la identificación y el tratamiento adecuado de valores ausentes o incompletos, asegurando que las decisiones tomadas respecto a estos datos sean las más adecuadas según el contexto de cada variable.

A pesar de que no se realiza un análisis, es muy importante realizar una buena limpieza en los datos, ya que el funcionamiento de la aplicación depende en si mismo de ello.

**Librerías utilizadas**

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.4.4     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.0
```

```
v purrr     1.0.2
-- Conflicts ---------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

### Cargado de datos

```
df_og <- read_csv("listings_bcn.csv")
```

```
Rows: 18086 Columns: 75
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (26): listing_url, source, name, description, neighborhood_overview, pi...
dbl  (37): id, scrape_id, host_id, host_listings_count, host_total_listings_...
lgl   (7): host_is_superhost, host_has_profile_pic, host_identity_verified, ...
date  (5): last_scraped, host_since, calendar_last_scraped, first_review, la...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

## Selección de variables

Seleccionaremos las variables que son relevantes para mostrarse en nuestro producto. Cuando son categóricas, luego las transformamos en factores.

```
df <- df_og |>
  select(id,
         name, # A pesar de que no es cualitativo, nos sirve para identificar al piso
         neighbourhood_group_cleansed,
         latitude,
         longitude,
         room_type,
         accommodates,
         bathrooms_text,
         bedrooms,
         beds,
         price,
         minimum_nights,
         maximum_nights,
         has_availability,
```

```
        availability_30,
        availability_60,
        availability_90,
        availability_365,
        number_of_reviews,
        number_of_reviews_ltm,
        number_of_reviews_l30d,
        review_scores_rating,
        review_scores_accuracy,
        review_scores_cleanliness,
        review_scores_checkin,
        review_scores_communication,
        review_scores_location,
        review_scores_value
        )
```

## Ajustes en las variables

En este apartado realizaremos cambios en el formato, tipo de datos u otros elementos de las variables de la base de datos.

### bathroom_text

Las categorías en bathroom_text son algo inconsistentes y a veces usan sinónimos para referirse a lo mismo. Modificaremos la columna para unificar los criterios y, a continuación, separar la columna en dos, una numérica y la otra booleana.

```
unique(df$bathrooms_text)
```

```
 [1] "2 baths"           "1.5 baths"         "2.5 baths"
 [4] "3 baths"           "1 shared bath"     "1 bath"
 [7] "1 private bath"    "3.5 baths"         "4 baths"
[10] "1.5 shared baths"  NA                  "2 shared baths"
[13] "2.5 shared baths"  "5.5 baths"         "7.5 baths"
[16] "4.5 baths"         "6 baths"           "0 shared baths"
[19] "Half-bath"         "Private half-bath" "0 baths"
[22] "5 baths"           "8 baths"           "3 shared baths"
[25] "8 shared baths"    "4 shared baths"    "Shared half-bath"
[28] "5 shared baths"    "3.5 shared baths"  "6 shared baths"
[31] "5.5 shared baths"  "10 baths"          "10 shared baths"
[34] "6.5 baths"         "4.5 shared baths"
```

```
df$bathrooms_text <- ifelse(df$bathrooms_text == "Shared half-bath", "0.5 Shared bath",
                            ifelse(df$bathrooms_text == "Private half-bath", "0.5 bath",
                            ifelse(df$bathrooms_text == "Half-bath", "0.5 bath", df$bathrooms

df$shared_bathrooms <- grepl("shared|Shared", df$bathrooms_text)

df$n_bathrooms <- sapply(strsplit(df$bathrooms_text, " "), function(x) as.numeric(x[1]))

bathrooms_text_position <- which(colnames(df) == "bathrooms_text")

df <- df |>
  relocate(all_of(c("n_bathrooms", "shared_bathrooms")), .before = bathrooms_text_position)
```

```
Warning: Using an external vector in selections was deprecated in tidyselect 1.1.0.
i Please use `all_of()` or `any_of()` instead.
  # Was:
  data %>% select(bathrooms_text_position)

  # Now:
  data %>% select(all_of(bathrooms_text_position))

See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
```

```
df <- df |>
  select(-bathrooms_text)

rm(bathrooms_text_position)
```

**price**

La variable "price" hace uso de símbolos de divisas distintas. Vamos a transformar el formato para poder cambiar la columna a tipo numérico.

```
df$price <- as.numeric(gsub("\\$", "", df$price))
```

```
Warning: NAs introduced by coercion
```

**review_score_rating**

Por diseño, una valoración no puede ser menor de 1. Por lo tanto, vamos a considerar todos los valores menores de 0 como NA's.

```r
df$review_scores_rating <- ifelse(df$review_scores_rating == 0, NA, df$review_scores_rating)
```

### Transformación de datos categóricos a factores

```r
df <- df |>
  mutate(
    neighbourhood_group_cleansed = factor(neighbourhood_group_cleansed),
    room_type = factor(room_type)
  )
```

### Renombre de columnas

```r
df_es <- df |>
  select(id,
         nombre = name,
         barrio = neighbourhood_group_cleansed,
         latitud = latitude,
         longitud = longitude,
         tipo_propiedad = room_type,
         capacidad = accommodates,
         banos = n_bathrooms,
         bano_compartido = shared_bathrooms,
         dormitorios = bedrooms,
         camas = beds,
         precio = price,
         noches_minimas = minimum_nights,
         noches_maximas = maximum_nights,
         disponible = has_availability,
         disponibilidad_30 = availability_30,
         disponibilidad_60 = availability_60,
         disponibilidad_90 = availability_90,
         disponibilidad_365 = availability_365,
         n_reviews = number_of_reviews,
         n_reviews_u30d = number_of_reviews_l30d,
         puntuacion_reviews_general = review_scores_rating,
         puntuacion_reviews_precision = review_scores_accuracy,
         puntuacion_reviews_limpieza = review_scores_cleanliness,
         puntuacion_reviews_checkin = review_scores_checkin,
         puntuacion_reviews_comunicacion = review_scores_communication,
         puntuacion_reviews_localizacion = review_scores_location,
         puntuacion_reviews_valor = review_scores_value
         )
```

### Ajuste de NA's

Hemos podido ver como todos los NA's encontrados son numéricos,

```
sumatorioNA <- df_es |> summarise(across(everything(), ~ sum(is.na(.))))


df_es <- df_es |>
  mutate(across(where(is.numeric), ~if (any(is.na(.))) floor(replace_na(., mean(., na.rm = TR
# df_es <- df_es |>
#   mutate(across(where(is.numeric), ~floor(replace_na(., mean(., na.rm = TRUE)))))
```

### Le damos orden a la tabla usando como referencia la variable id

```
df_es <- df_es |>
  arrange(df_es, id)
```

## Guardado de archivo

Almacenamos los datos modificados para poder ingresar los directamente en la aplicación de Shiny o en C++

Formato RDS

```
saveRDS(df_es, "clean_airbnb_data.rds")
```

Formato CSV con las columnas latitud y longitud, para poder hacer el análisis KDE

```
df_kde <- df_es |>
  select(latitud, longitud)
```

```
write.csv(df_kde, "codigo_cpp/airbnb.csv", row.names = FALSE)
```

## Aplicación de Shiny

Las aplicaciones de Shiny suelen constar de dos partes diferenciadas: La interfaz o IU y la parte del server. La primera es la que indica la posición de los elementos y la segunda la forma de interactuar de estos. A continuación, adjunto el código del dashboard con comentarios para explicarlo.

```r
library(shiny)
library(DT)
```

```
Attaching package: 'DT'

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

    dataTableOutput, renderDataTable
```

```r
library(MASS)
```

```
Attaching package: 'MASS'

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

    select
```

```r
# Cargamos el archivo RDS
df_shiny <- readRDS("clean_airbnb_data.rds")

# Ajustes para el filtrado, necesarios para que no aparezcan números en las listas de opci
df_shiny$barrio <- as.character(df_shiny$barrio)
df_shiny$tipo_propiedad <- as.character(df_shiny$tipo_propiedad)

# UI
ui <- fluidPage(
  titlePanel("Airbnb Apartment Dashboard"),

  sidebarLayout(# Apartado de la izquierda
    sidebarPanel(

      h1("Menú de filtros"),

      br(),

      h2("Localización"),
      selectInput("barrio", "Barrio", choices = c("Todos", unique(df_shiny$barrio))),
      sliderInput("latitud", "Latitud", value = c(min(df_shiny$latitud, na.rm = TRUE), max(d
```

```
    sliderInput("longitud", "Longitud", value = c(min(df_shiny$longitud, na.rm = TRUE), max

    br(),

    h2("Características de la propiedad"),
    selectInput("tipo_propiedad", "Tipo de propiedad", choices = c("Todos", unique(df_shiny
    sliderInput("capacidad", "Capacidad de personas", value = c(min(df_shiny$capacidad, na
    sliderInput("banos", "Número de baños", value = c(min(df_shiny$banos, na.rm = TRUE), ma
    selectInput("bano_compartido", "Baño compartido", choices = c("Todos", unique(df_shiny$
    sliderInput("dormitorios", "Número de dormitorios", value = c(min(df_shiny$dormitorios
    sliderInput("camas", "Número de camas", value = c(min(df_shiny$camas, na.rm = TRUE), ma

    br(),

    h2("Condiciones de la reserva"),
    sliderInput("precio", "Precio", value = c(min(df_shiny$precio, na.rm = TRUE), max(df_sh
    sliderInput("noches_minimas", "Noches mínimas", value = c(min(df_shiny$noches_minimas,
    sliderInput("noches_maximas", "Noches máximas", value = c(min(df_shiny$noches_maximas,

    br(),

    h2("Disponibilidad"),
    selectInput("disponible", "¿Está disponible?", choices = c("Todos", unique(df_shiny$dis
    numericInput("disponibilidad_30", "Mínimo de disponibilidad en los próximos 30 días", v
    numericInput("disponibilidad_60", "Mínimo de disponibilidad en los próximos 60 días", v
    numericInput("disponibilidad_90", "Mínimo de disponibilidad en los próximos 90 días", v
    numericInput("disponibilidad_365", "Mínimo de disponibilidad en los próximos 365 días"

    br(),

    h2("Reseñas"),
    sliderInput("puntuacion_reviews_general", "Puntuación general de reseñas", value = c(mi
    sliderInput("puntuacion_reviews_precision", "Puntuación de reseñas en precisión", value
    sliderInput("puntuacion_reviews_limpieza", "Puntuación de reseñas en limpieza", value =
    sliderInput("puntuacion_reviews_checkin", "Puntuación de reseñas en check-in", value =
    sliderInput("puntuacion_reviews_comunicacion", "Puntuación de reseñas en comunicación"
    sliderInput("puntuacion_reviews_localizacion", "Puntuación de reseñas en localización"
    sliderInput("puntuacion_reviews_valor", "Puntuación de reseñas en valor", value = c(min

  ),
  mainPanel(# Panel principal en el centro del dashboard
    h1("Visualización KDE"),
```

```
      plotOutput("kdePlot"),
      h1("Tabla filtrada"),
      DTOutput("tablaFiltrada")
    )
  )
)

# Server
server <- function(input, output) {

  observe({
    # Filtrado con dplyr con todas las posibilidades de filtrados incluidas en el código y su
    filtered_data <- df_shiny |>
      filter(if(input$barrio != "Todos") barrio == input$barrio else TRUE) |>
      filter(if(input$tipo_propiedad != "Todos") tipo_propiedad == input$tipo_propiedad else
      filter(if(input$disponible != "Todos") disponible == (input$disponible == "Sí") else TR
      filter(latitud >= input$latitud[1], latitud <= input$latitud[2]) |>
      filter(longitud >= input$longitud[1], longitud <= input$longitud[2]) |>
      filter(capacidad >= input$capacidad[1], capacidad <= input$capacidad[2]) |>
      filter(banos >= input$banos[1], banos <= input$banos[2]) |>
      filter(dormitorios >= input$dormitorios[1], dormitorios <= input$dormitorios[2]) |>
      filter(camas >= input$camas[1], camas <= input$camas[2]) |>
      filter(precio >= input$precio[1], precio <= input$precio[2]) |>
      filter(noches_minimas >= input$noches_minimas[1], noches_minimas <= input$noches_minima
      filter(noches_maximas >= input$noches_maximas[1], noches_maximas <= input$noches_maxima
      filter(disponibilidad_30 >= input$disponibilidad_30) |>
      filter(disponibilidad_60 >= input$disponibilidad_60) |>
      filter(disponibilidad_90 >= input$disponibilidad_90) |>
      filter(disponibilidad_365 >= input$disponibilidad_365) |>
      filter(puntuacion_reviews_general >= input$puntuacion_reviews_general[1], puntuacion_re
      filter(puntuacion_reviews_precision >= input$puntuacion_reviews_precision[1], puntuacio
      filter(puntuacion_reviews_limpieza >= input$puntuacion_reviews_limpieza[1], puntuacion_
      filter(puntuacion_reviews_checkin >= input$puntuacion_reviews_checkin[1], puntuacion_re
      filter(puntuacion_reviews_comunicacion >= input$puntuacion_reviews_comunicacion[1], pun
      filter(puntuacion_reviews_localizacion >= input$puntuacion_reviews_localizacion[1], pun
      filter(puntuacion_reviews_valor >= input$puntuacion_reviews_valor[1], puntuacion_review

    # Verificamos si se han producido NA's en el proceso anterior
    if(nrow(filtered_data) > 0) {

      # Realizar el análisis KDE con el paquete
      kde_result <- MASS::kde2d(filtered_data$longitud, filtered_data$latitud, n = 100)
```

```
      kde_data <- expand.grid(x = kde_result$x, y = kde_result$y)
      kde_data$z <- as.vector(kde_result$z)

      # ggplot con los puntos y el scale gradient
      ggplot_data <- ggplot() +
        geom_tile(data = kde_data, aes(x = x, y = y, fill = z), alpha = 0.8) +
        geom_point(data = filtered_data, aes(x = longitud, y = latitud)) +
        scale_fill_gradient(low = "white", high = "red") +
        labs(fill = "Densidad KDE")

      # Renderizar gráfico KDE en ggplot
      output$kdePlot <- renderPlot({
        ggplot_data
      })
    }

    # Renderizar la tabla filtrada
    output$tablaFiltrada <- renderDT({
      datatable(filtered_data)
    })
  })
}

# Ejecuta la aplicación
shinyApp(ui = ui, server = server)
```

## Uso de otros lenguajes de programación

Tal y como hemos visto anteriormente, podemos combinar lenguajes de programación, de forma que utilicemos el lenguaje más conveniente en todo momento según el proceso que se esté realizando.

Algunos procesos dentro de la ciencia de datos en los que se combinan lenguajes de programación son los siguientes:

- SQL y Python/R para el manejo y análisis de datos al tener que consultar directamente en una base de datos relacional para poder extraer datos para análisis estadístico o de machine learning.
- Python/R y Javascript para crear visualizaciones interacivas en la web.
- Python/R y C/C++ para realizar tareas de alto rendimiento

- Python/R y la terminal para automatizar tareas y flujo de trabajo.
- Python/R y Hadoop Spark cuando se este trabajando con un volumen de datos elevado que no quepa en una máquina.

En la ciencia de datos, ser capaz de combinar lenguajes de programación aporta mucho valor y abre un abanico de posibilidades a la hora de realizar las tareas propias de esta rama.

Para nuestro trabajo, hemos realizado el proceso del KDE en C++, con la idea de implementarlo directamente en el workflow de la aplicación Shiny. Sin embargo, hemos terminado descartando esta opción, ya que las funciones en R de KDE han acabado siendo mas eficientes en este caso.

El código utilizado para C++ lo comentaremos a continuación:

```cpp
#include <iostream> //Biblioteca de inputs y outputs de C++
#include <fstream> //Biblioteca para interactuar con archivos de tipo texto
#include <vector> //Biblioteca que incorpora la estructura del vector
#include <cmath> //Biblioteca que incluye funciones matemáticas
#include <sstream> //Biblioteca para trabajar con los flujos de strings
#include <iomanip> //Biblioteca que incluye funciones para formatear salidas de datos

using namespace std; // Estandar no tener que usar el prefijo std:: en funciones de entradas

// Función para calcular KDE Gaussiano. Toma tres parámetros, las dos variables geográficas y
vector<double> kdeGaussiano(const vector<double>& x, const vector<double>& data, double bandw
    int n = x.size(); // Almacena el tamaño de x
    int m = data.size(); // Obtiene el tamaño del vector
    vector<double> output(n); // Crea un vector del mismo tamaño que x

    // Fórmula KDE
    for (int i = 0; i < n; i++) { //Itera sobre los elementos de x
        double sum = 0; //Inicializa una suma para calcular el KDE
        for (int j = 0; j < m; j++) {
            double u = (x[i] - data[j]) / bandwidth;
            sum += exp(-0.5 * u * u) / sqrt(2 * M_PI);
        }
        output[i] = sum / (m * bandwidth); //Calcula el valor de KDE para el punto i y lo alm
    }

    return output; //Retorna el vector resultante
}

int main() {
```

```
  //Define los nombres de los archivos de entrada y salida
    const string inputFileName = "airbnb.csv";
    const string outputFileName = "airbnb_result.csv";
    const double bandwidth = 0.1;  // Ajusta el ancho de banda según tus necesidades

    ifstream inputFile(inputFileName);
    ofstream outputFile(outputFileName);
// Condicionante por si el archivo input no existe
    if (!inputFile) {
        cerr << "Error al abrir el archivo " << inputFileName << endl;
        return 1;
    }

    // Lee los datos del archivo CSV
    vector<double> latitudes;
    vector<double> longitudes;
    string line;
    // Mantiene funcionando el programa mientras queden entradas por analizar.
    while (getline(inputFile, line)) {
        istringstream iss(line);
        double latitude, longitude;
        char comma;
        iss >> latitude >> comma >> longitude;
        latitudes.push_back(latitude);
        longitudes.push_back(longitude);
    }

    // Aplicar el análisis KDE a las latitudes
    vector<double> kdeLatitudes = kdeGaussiano(latitudes, latitudes, bandwidth);

    // Escribir los resultados en el archivo de salida
    for (int i = 0; i < latitudes.size(); i++) {
        outputFile << fixed << setprecision(6) << latitudes[i] << "," << longitudes[i] << ","
    }

    cout << "Proceso completado: " << outputFileName << endl;

    inputFile.close();
    outputFile.close();

    return 0;
}
```

A diferencia del programa en R, el tiempo de ejecución era algo más rápido, teniendo en cuenta de que los datos se leían directamente de un CSV y se incorporaban en otro.

## Aplicación de principios de buen diseño

A continuación vamos a analizar y argumentar por qué las dos aplicaciones (R y C++) tienen un buen diseño o en que pueden mejorar

### Interfaz en R

La estructura del código está bien definida, utiliza bibliotecas relevantes y son activadas en la parte superior y hay una separación clara entre la parte de ui y el server, que es la estructura de Shiny.

Por otro lado, el formato escogido para cargar el conjunto de datos, RDS, es el más apropiado para exportar e importar entre sesiones de R.

En la parte de UI podemos ver como se divide la interfaz entre la zona principal y el "sidebar". Los controles de filtrado son adecuados, ya que permiten marcar los mínimos y máximos cuando se puede requerir y seleccionar opciones en las variables categóricas.

En cuanto a la parte del server, el filtrado de datos utiliza el formato tidyverse lo cual lo hace más claro al lector. También se hacen verificaciones NA's y de no insertar conjuntos de datos vacíos. A la hora de visualizar datos se utilizan buenos diseños y con colores adecuados.

En cuanto al cumplimiento de principios SOLID y de buen diseño en sí, podemos ver como cada función en el código tiene una responsabilidad única, y está estructurado con la posibilidad de poder ser extendido.

Se podrían mejorar varios elementos, como el incremento de comentarios, aumentar el manejo de excepciones y hacer filtrados más eficientes, además de aprovechar la capacidad de añadir interfaces interactivas.

### Programa C++

El código C++ proporcionado demuestra una implementación eficiente para el análisis de Estimación de Densidad del Kernel (KDE) en un conjunto de datos, utilizando adecuadamente las capacidades del lenguaje.

En cuanto a los principios SOLID, el código refleja parcialmente estos conceptos. La función kdeGaussiano ilustra el principio de Responsabilidad Única al centrarse en una sola tarea, que es calcular el KDE.

Aunque la extensibilidad (principio Abierto/Cerrado) está presente en cierta medida, por ejemplo, en la forma en que se podría extender kdeGaussiano para diferentes métodos de KDE sin alterar su estructura actual, los principios de Sustitución de Liskov, Segregación de Interfaces e Inversión de Dependencias no son directamente aplicables o evidentes en este fragmento, principalmente porque el código no emplea una estructura orientada a objetos con herencia o interfaces, que son más relevantes para estos principios.

El diseño del código muestra claridad y modularidad, con una buena separación entre el procesamiento de datos y las operaciones de entrada/salida.

Las funciones y variables están nombradas de manera descriptiva, lo que facilita la comprensión del propósito de cada segmento del código.

El manejo de archivos es adecuado, asegurando que los recursos se abran y cierren apropiadamente.

En términos de mejora, el código podría beneficiarse de una mayor abstracción o incluso de una reestructuración orientada a objetos para alinearse más estrechamente con todos los principios SOLID, aunque esto dependería del contexto y de los requisitos más amplios del proyecto.

## Estructuras dinámicas

Las estructuras dinámicas en programación son aquellas que pueden cambiar de tamaño o forma durante la ejecución del programa, adaptándose a las necesidades de los datos en tiempo real.

Por ejemplo, en C++ las estructuras son fundamentales para manejar datos de manera flexible y eficiente.

Podemos encontrar un ejemplo de estructuras dinámicas en nuestro código de C++. Los vectores “latitudes”, “longitudes” y “kdeLatitudes” son ejemplos de estructuras dinámicas.

Pese a que en nuestro código la implementación de estructuras dinámicas es adecuada. Podríamos expandir sus funcionalidades. Algunas formas son las siguientes:

**Reserva de espacio en memoria**

Para elementos que pueden cambiar de forma como el vector kdeLatitudes, podemos mejorar la eficiencia reservando espacio en memoria con un rango aproximado.

**Uso de punteros**

Si el uso de memoria hubiera sido mas exigente habría sido buena práctica aumentar el uso de punteros.

**Estructuras de datos dinámicas**

Si hubiera tenido que hacer uso de estructuras más complejas, podría haber hecho uso de otro tipo de estructuras como maps o tuples.

## Análisis de resultados

Podemos determinar que los objetivos propuestaos se han cumplido, en primer lugar la herramienta es funcional, funciona con fluidez y hace uso de datos reales de una población completa. Con la ayuda del KDE proporcionamos visualmente mucha información, identificando inmediatamente las áreas de alta demanda o saturación de mercado.

Los resultados que nos ofrece el programa son relevantes para los posibles demandantes de hacer uso de un programa así y pueden ser una base sólida para tomar decisiones estratégicas.