Modelo de optimización de asignación de ejecutivos a gerentes

Tabla de contenidos

Optimización 4
Funcionamiento 4
Factibilidad 4
Selección, combinación y mutación 5
Implementación 5
Configuración 6
Paquetes 6
Lectura de datos 6
Funciones 6
genera_solucion 6
mezcla_soluciones 7
evalua_solucion 8
iniciar 10
iterar 10
recorrer_iteraciones 12
optimizar_asignaciones 12
Despliegue 14
Implementación 15
Ejecución 15
Guardado 16
Conclusión 19

Introducción

La solución del problema de capacidad consiste en la asignación óptima de los ejecutivos de banco a los gerentes tomando en consideración las siguientes restricciones:

- Se cuenta en total con una lista de Gerentes Comerciales, Ejecutivos y Clientes. Los clientes están asignados a ejecutivos, y los ejecutivos se asignan a gerentes. Los clientes no pueden cambiar de ejecutivo al cual se encuentran asignados.
- A los gerentes se le puede asignar cualquier ejecutivo siempre y cuando el ejecutivo y el gerente sean de la misma zona.
- Solo se le debe asignar el ejecutivo a un gerente, no es posible asignar un ejecutivo a varios gerentes.
- Los ejecutivos se asignan a los gerentes, y todos los clientes del ejecutivo se asignan al gerente (No se puede asignar la mitad de los clientes de un ejecutivo a un gerente, es todo o nada).
- Los clientes no pueden moverse ni reasignarse a otros ejecutivos.
- No debe superarse el tiempo disponible de cada gerente, teniendo en cuenta las actividades que ejecutará durante el año con los clientes asignados. Esto es de acuerdo con los productos y actividades comerciales de los clientes que se asignen.
- Los clientes están clasificados como tipo A, B y C donde la prioridad está dada en ese mismo orden.
- Se debe tener en cuenta la variable score de la base clientes, se le debe asignar a los gerentes los ejecutivos con los scores de clientes más altos. Es posible que no todos los clientes se puedan asignar.

La métrica para evaluar este modelo se basa en el número de clientes asignados a los gerentes, el número de clientes por fuera de la asignación y su prioridad.

Esta métrica está dada por la siguiente relación:

$$\min_{s} f(s) = \frac{a_s + b_s}{r_s}$$

donde:

s: es la asignación de los ejecutivos a los gerentes.

a_s: es la cantidad de clientes tipo A asignados en la asignación s.

b_s: es la cantidad de clientes tipo B asignados en la asignación s.

r_s: es la cantidad de clientes sin asignar.

Optimización

La estrategia de optimizacion seleccionada para abordar el problema consiste en un proceso de algoritmo evolutivo. Este se realiza con el fin de seleccionar la asignación óptima de entre un conjunto de asignaciones. A continuación se explican sus detalles.

Funcionamiento

El algoritmo evolutivo planteado presenta el siguiente funcionamiento:

- Se genera una familia aleatoria de asignaciones:
- Hasta determinar convergencia:
  - Se evalúan las asignaciones presentadas.
  - Se dividen en dos grupos:
    - Las mejores, que se conservan para la siguiente generación.
    - Las viejas, que se conservan aleatoriamente para la siguiente generación.
  - Se construye una nueva generación de la siguiente manera:
    - Se conservan las mejores asignaciones.
    - Se conservan las asignaciones viejas
    - Se construyen unas asignaciones nuevas. Estas se generan a partir de las anteriores mediante unas reglas de combinación.
  - A medida que se van creando nuevas generaciones, los procesos de recombinación, selección y mutación optimizan su desempeño con respecto a la función objetivo.
- Se selecciona la mejor asignación de la generación final.

A grandes rasgos, este es el algoritmo seleccionado e implementado para la solución del problema. No obstante, resulta interesante profundizar en algunos detalles.

Factibilidad

Debido a las restricciones del problema, no todas las asignaciones resultan adecuadas. No obstante, durante el proceso de desarrollo resulta evidente que se evidenció que la generación aleatoria de asignaciones factibles representa una carga computacional que resulta inconveniente.

Con el fin de abordar este problema se dispuso dentro de la función objetivo un factor de factibilidad, que entrega mejores valores a medida que las asignaciones son más factibles. De esta forma, las asignaciones van generando su propia factibilidad a medida que se iteran las generaciones.

Esta estrategia evidenció un buen comportamiento. En la implementación fue posible observar que las asignaciones eran factibles de una generación en adelante.

Selección, combinación y mutación

Los algoritmos evolutivos se caracterizan por incluir pasos de selección, combinación y mutación.

- Para la selección se plantea la permanencia del mejor 2% de la población de una generación a la siguiente. Esto garantiza que, en caso de encontrar asignaciones con valores muy favorables, estas no serán olvidadas en el ejercicio a menos que sean encontradas otras mejores.

- Para el proceso de combinación entre asignaciones, se propone una combinación por parejas. Para esto se realiza una selección aleatoria entre los elementos de ambas asignaciones. En esto es necesario tomar en consideración las restricciones.

- El proceso de mutación se realiza de dos maneras. La primera de ellas consiste en la inclusión forzada de una asignación aleatoria en el proceso de combinación. Esto modifica el proceso de combinación, generando asignaciones cuyos elementos se encuentran aleatorizados. La segunda consiste en la inclusión forzada de una asignación vacía en el proceso de combinación. En esta asignación vacía los ejecutivos no se encuentran asignados a ningún gerente. modifica el proceso de combinación, generando asignaciones donde varios ejecutivos que estaban asignados en la generación anterior, no lo están en la nueva.

Implementación

Si bien el código no es óptimo por factores de tiempo, es posible mencionar algunos detalles de la implementación.

- La implementación se realiza bajo el paradigma de programación funcional. La solución presentada está dividida por módulos funcionales independientes que pueden incluso ser reutilizados en implementaciones posteriores.

- Para optimizar el uso de recursos computacionales, el algoritmo evolutivo se ejecuta de forma paralela utilizando varios hilos. Esto facilita la realización de cálculos complejos de manera simultánea y disminuye los tiempos de respuesta de la máquina.

Configuración

Para la solución del problema propuesto se realiza inicialmente la configuración del entorno de desarrollo adecuado para el desarrollo y prueba de los scripts de transformación y modelamiento de datos que haya lugar.

Paquetes

La configuración del entorno requiere la disposición del software correspondiente. Esto se realiza mediante el llamado de paquetes especializados para tareas iterativas.

```
library("readr")
library("purrr")
library("tidyr", exclude = "extract")
library("magrittr")
library("dplyr")
library("tictoc")
library("parallel")
```

Lectura de datos

Los datos son cargados para el correcto desarrollo del modelo de optimización. Si bien es necesario cargar los datos procesados por el código anterior, también se realiza una consulta para conocer el número total de clientes por región.

```
read_rds("01_datos/tabla_ejecutivo_tiempo_region_marca.rds") ->
tabla_ejecutivo_tiempo_region_marca
read_rds("01_datos/tabla_gerente_tiempo_region.rds") ->
tabla_gerente_tiempo_region

read_csv(
  "01_datos/pcac_mac_gpi_clientes.csv"
  ) -> pcac_mac_gpi_clientes

pcac_mac_gpi_clientes %>%
  count(cod_region_gte_inv) -> clientes_region
```

Funciones

A continuación se muestran las funciones desarrolladas con el fin de implementar la estrategia de optimización seleccionada.

genera_solucion

Esta función genera una asignación aleatoria, que puede o no ser factible. Se utiliza en la construcción de la primera generación de asignaciones.

```
genera_solucion <- function(elemento, tab_ejec_t_m, tab_ger_t){
  sample_n(tab_ger_t, nrow(tab_ejec_t_m), replace = TRUE) %>%
    bind_cols(tab_ejec_t_m)
}
```

Este código hace uso de la tabla de ejecutivos y la tabla de gerentes. Con estas tablas cargadas realiza los siguientes pasos:

- Toma una muestra aleatoria de la tabla de gerentes con reemplazo, del tamaño de la tabla de ejecutivos.
- Anexa la tabla aleatoria de gerentes a la tabla de ejecutivos. Con esto cada ejecutivo queda asignado a un gerente.

mezcla_soluciones

La siguiente función realiza el proceso de combinar dos asignaciones. En este proceso se toma en consideración el paso de mutación mencionado anteriormente.

```
mezcla_soluciones <- function(sol_1, sol_2, mutacion){

  roll_the_dice <- c(0.5, 0.5, 0)

  if(runif(1) < 0.05){
    roll_the_dice <- c(0.4, 0.4, 0.2)
    }

  if(runif(1) < 0.15){
    new_order <- sample(seq_len(nrow(sol_1)))
    sol_1 %>%
      mutate(
        cod_gte_inv = cod_gte_inv[new_order],
        tiempo_restante = tiempo_restante[new_order]
      ) -> sol_1
    }

  sample(
    c(1, 2, 3),
    nrow(sol_1),
    replace=TRUE,
    prob = roll_the_dice
    ) -> ind
  bind_rows(
    filter(sol_1, ind == 1),
    filter(sol_2, ind == 2),
    filter(mutacion, ind == 3),
  ) %>% arrange(cod_ejec_bco)

}
```

El código presentado hace uso de la primera asignación, de la segunda asignación y de la tabla vacía de mutación. Esta última corresponde a la mutación mencionada donde los ejecutivos no se relacionan con ningún gerente. Con estos insumos se realizan las siguientes tareas:

- Generar un aleatorio con el fin de incluir esta tabla vacía dentro de la aleatorización, En la primera opción (95% de las veces), las asignaciones se combinan en proporción 50 - 50, en la segunda opción (5% de las veces) las asignaciones se combinan en proporción 40 - 40 dejando un 20% de espacio para ejecutivos no gerenciados. Esto constituye una mutación que retira ejecutivos forzosamente para que se vuelvan a asignar de una manera distinta más adelante.

- Generar otro aleatorio con el fin de reemplazar forzosamente la primera solución por una solución falsa, generada aleatoriamente. Esto constituye una mutación que reemplaza la asignación 1 con otra desordenada, rompiendo la relación de los gerentes y los ejecutivos para encontrar nuevas relaciones posibles.

- Generar un vector aleatorio de tres entradas, 1, 2 y 3, que se utilizarán para escoger los elementos de las asignaciones `sol_1`, `sol_2` y `mutación`. En la construcción de este vector aleatorio se hace uso de las probabilidades antes mencionadas.

- Conformar una nueva asignación tomando registros de la primera, de la segunda y de ser el caso, de la mutación. Con esto se combinan dos asignaciones, generando una asignación hija que hará parte de la nueva generación.

evalua_solucion

La función de evaluación tiene como objetivo establecer un puntaje para cada una de las asignaciones. Este puntaje es calculado de acuerdo con la función a maximizar presentada en la sección anterior.

```
evalua_solucion <- function(sol, objetivo){

    sol %$% sum(clientes) -> cli_tot

  sol %>%
    group_by(cod_gte_inv) %>%
    summarise(
      tiempo_restante = mean(tiempo_restante),
      marca_a = sum(marca_a),
      marca_b = sum(marca_b),
      clientes = sum(clientes),
      tiempo_usado = sum(tiempo_x_ejecutivo)
    ) -> tabla_gerentes
```

```
  tabla_gerentes %>%
    filter(tiempo_restante > tiempo_usado) %$%
    list(a = sum(marca_a), b = sum(marca_b), r  = cli_tot -
sum(clientes), g = nrow(.), f = (nrow(.) + 1  == nrow(tabla_gerentes))) -
> asignados

  with(asignados,  - (a + b)/(r + 0.001) - g - 100*f)

}
```

El código presentado se encarga de establecer el puntaje de una asignación entre ejecutivos y gerentes mediante los siguientes pasos.

- Suma todos los clientes. Este paso se puede mejorar calculando esta suma de manera externa a la función. No obstante, al hacerlo, la función aumenta sus dependencias y por ende pierde autonomía y dificulta su posible uso en otro contexto.
- Agrupa la asignación actual por gerente.
- Calcula el tiempo restante por gerente, la cantidad de clientes de tipo A, la cantidad de clientes de tipo B, el total del tiempo usado y la cantidad de tiempo requerido. Guardando estos cálculos en una tabla nueva.
- Filtra los gerentes cuyo tiempo requerido es menor que el tiempo disponible. Con esto se garantiza la evaluación sobre la parte factible de la asignación.
- Realiza los cálculos de totales para los clientes tipo A (a), tipo B (b), y los restantes (r). Adicionalmente calcula el número de gerentes asignados (g) y si la asignación completa es factible (f).
- Estos valores son utilizados en la fórmula para el puntaje de la asignación. La fórmula tiene dos partes: la primera corresponde a la optimización realizada, la segunda consiste en un puntaje de factibilidad que premia las asignaciones factibles.

$$\min_s f_0(s) = \underbrace{\frac{a_s + b_s}{r_s + 0.001}}_{\text{optimizacion}} + \underbrace{g + 100f}_{\text{factibilidad}}$$

Una corrección importante realizada a la fórmula de optimización corresponde a la suma de una milésima en el numerador. Este cambio premia con mil puntos las asignaciones que no presentan clientes sin ejecutivo.

iniciar

La función de iniciar el algoritmo hace uso de las funciones definidas anteriormente para generar una primera familia de asignaciones de un tamaño determinado. Este grupo de asignaciones constituye la primera generación.

```
iniciar <- function(tab_ejec, tab_ger, n_ind, vec_obj){

  tibble(
    elementos = seq_len(n_ind),
    soluciones = map(elementos, genera_solucion, tab_ejec, tab_ger),
    evaluacion = map_dbl(soluciones, evalua_solucion, vec_obj)
  )

}
```

El código presentado utiliza una tabla de ejecutivos, una tabla de gerentes y el número de individuos a generar. Incluye también un comodín para futuras correcciones. Con estos elementos se realizan las siguientes acciones.

- Define una tabla en la que integra un número para cada asignación.
- Genera soluciones y las guarda como una columna de esta tabla.
- Evalúa las soluciones que recién ha creado. Los valores correspondientes a las evaluaciones los guarda en la misma tabla.
- Retorna la tabla con las tres columnas mencionadas.

iterar

La función de iteración conforma el paso que se repite hasta encontrar el óptimo. Consiste en tomar una tabla de soluciones y por medio de operaciones obtener una tabla nueva. La tabla insumo corresponde a la generación actual y la tabla generada corresponde a la generación siguiente. Esta función implementa los conceptos de selección, combinación y mutación mencionados.

Su diseño está orientado a mejoras incrementales relativamente pequeñas entre una generación y la siguiente. Esto conlleva un aumento en el número de iteraciones necesarias para encontrar un óptimo. No obstante, tiene dos ventajas: permite una exploración más detallada de las diferentes posibilidades de solución y facilita el trabajo con poblaciones grandes sin un consumo computacional excesivo.

```
iterar <- function(tabla_sol, n_ind, tabla_sin_gerencia){

  bunch <- floor(n_ind/50)

  tabla_sol %>%
  slice_min(
```

```
    n = bunch,
    order_by = evaluacion,
    with_ties = FALSE
    ) -> best_people

  tabla_sol %$%
    tibble(
      elementos = seq_len(bunch*4),
      soluciones_1 = sample(soluciones, bunch*4),
      soluciones_2 = sample(soluciones, bunch*4),
      soluciones = map2(soluciones_1, soluciones_2, mezcla_soluciones,
tabla_sin_gerencia),
      evaluacion = map_dbl(soluciones, evalua_solucion, vec_obj)
    ) -> new_people

  tabla_sol %>%
    sample_n(bunch*45) -> old_people

  bind_rows(best_people, new_people, old_people) %>%
    arrange(evaluacion) -> new_sol

  traza <<- c(
    traza,
    list(new_sol[["evaluacion"]])
  )

  new_sol

}
```

El código presentado toma como insumo la tabla actual de asignaciones, el número de individuos y la tabla ejecutivos no asignados. A partir de allí, la función efectúa los siguientes pasos.

- Define paquetes en los datos, para trabajar fácilmente. Cada paquete tiene un peso del 2%.
- Recupera el mejor 2% de todos los registros y lo guarda en un primer paquete. Este es el paso de selección en el algoritmo.
- Selecciona aleatoriamente dos segmentos, en cada uno el 8% de la población.
- Combina los segmentos utilizando la función presentada anteriormente. Este paso contiene los procedimientos de combinación y mutación del algoritmo. El resultado es un nuevo conjunto de asignaciones, cuyo tamaño es el 8% de la población.
- Evalúa este nuevo segmento, otorgando puntajes a los nuevos individuos.

- El 90% restante permanece sin cambios para la siguiente generación.

recorrer_iteraciones

La función de recorrer iteraciones se encarga de repetir la función anterior un número determinado de veces. En la construcción del presente prototipo se requirió muchas veces probar con distintas longitudes de iteración. Por lo tanto no se implementó un criterio de parada.

```
recorrer_iteraciones <- function(n_iter, tabla_ini, n_ind, tabla_mutante)
{
  tabla_sol <- tabla_ini
  for(i in seq_len(n_iter)){
    tabla_sol <- iterar(tabla_sol, n_ind, tabla_mutante)
    }
  tabla_sol
}
```

Esta función toma como insumos, el número de iteraciones, la tabla inicial, el número de individuos y la tabla de asignaciones vacías. Con estos elementos se efectúan las siguientes operaciones:

- Generación de una copia de la tabla inicial para iterar.
- Repetición de los siguientes pasos en bucle hasta que el número de iteraciones acabe.
  - Se calcula la nueva generación a partir de la función iterar presentada.
  - Se reemplaza la generación anterior con la nueva.
- Entrega de la última generación.

optimizar_asignaciones

La función optimizar asignaciones se encarga de unir las funciones anteriores y facilitar su empleo.

```
optimizar_asignaciones <- function(lista_pareja_tablas, n_pob, niter){

  lista_pareja_tablas[["tab_ejec"]] -> tab_ejec_reg
  lista_pareja_tablas[["tab_ger"]] -> tab_ger_reg

  tibble(0,0, .name_repair = "minimal") %>%
    set_names(
      names(tab_ger_reg)
    ) -> gerente_muy_ocupado

  genera_solucion(0, tab_ejec_reg, gerente_muy_ocupado) ->
```

```
tabla_sin_asignar

  iniciar(
    tab_ejec_reg,
    tab_ger_reg,
    n_pob*1.5, meta
    ) -> inicial

  recorrer_iteraciones(niter, inicial, n_pob, tabla_sin_asignar) ->
optimus

  optimus

}
```

El código presentado requiere para su funcionamiento una pareja de tablas, de gerentes y ejecutivos, guardada convenientemente en la forma de una lista; el número de individuos por generación y el número de iteraciones a utilizar. Estos dos últimos resultan muy importantes, pues al no haber un criterio de parada, deben ser manipulados por el usuario. Este manejo debe ser cuidadoso, pues define completamente la duración y el uso de recursos computacionales del algoritmo. Muchas iteraciones o generaciones demasiado grandes pueden conllevar problemas de máquina.

Con esto claro, los pasos que se siguen para la optimización son:

- Guardar las tablas de ejecutivos y de gerentes en objetos distintos para su uso.
- Construir una tabla con un nuevo gerente. Su tiempo disponible y su código de gerente son iguales 0.
- Por medio de la función que genera las soluciones se crea una asignación donde todos los ejecutivos son asignados a este gerente nuevo. Todos los ejecutivos de esta tabla no están asignados a un gerente real. Mediante esta configuración es posible tener un espacio de no asignación. Es decir, mientras los ejecutivos se encuentren asignados al gerente falso se consideran sin asignar.
- Se genera la población inicial, 1.5 veces más grande que la requerida, con esto se garantiza mayor variedad en el proceso de selección aleatoria.
- El proceso iterativo se realiza mediante la función programada para este fin. Las iteraciones se repiten el número especificado y se entrega una tabla de asignaciones final, en la que se encuentra la última generación producida y por ende, el óptimo.

Despliegue

Para desplegar la solución elaborada es suficiente con disponer de una máquina virtual cuya instalación de docker se encuentre actualizada. El presente proyecto se puede ejecutar al interior de un contenedor `rocker/shiny-verse` mediante un cron desatendido periódico, con una frecuencia mensual o quincenal según estime la organización.

La asignación resultante se puede disponibilizar a través de un servicio web o directamente en un portal de capacidad.

Para realizar estas tareas exitosamente es necesario contar con una estrategia de integración continua que permita iterar las rutinas de código y corregir factores críticos.

Es posible que sea necesario disminuir los hilos de la computación paralela al implementar el contenedor. La lectura de los datos debe hacerse vía consultas a bases sql del sistema de información del área. Esto también puede requerir un esfuerzo para la implementación de cambios en la alimentación del modelo.

De manera que, la integración del modelo de optimización presentado se realiza sin mayores inconvenientes.

Implementación

Terminando la carga de los datos, las funciones y los paquetes, se procede a la implementación del algoritmo.

Ejecución

Para ello se disponen las tablas de ejecutivos y gerentes que son introducidas en el algoritmo con un total de 1000 iteraciones sobre generaciones de 2000 individuos. El uso de computación paralela permite trabajar los datos de todas las regiones de manera simultánea.

```
tabla_ejecutivo_tiempo_region_marca %>%
  nest(tab_ejec = c(-cod_region_ejec_bco)) -> ejecutivos_region

tabla_gerente_tiempo_region %>%
  nest(tab_ger = c(-cod_region_gte_inv)) -> gerentes_region

traza <- list()

tic()
inner_join(
  ejecutivos_region, gerentes_region,
  by = c("cod_region_ejec_bco" = "cod_region_gte_inv")
  ) %>%
  mutate(
    pares_tablas =
      transpose(
        list(tab_ejec = tab_ejec, tab_ger = tab_ger)
        ),
    optimos =
      mclapply(
        pares_tablas,
        optimizar_asignaciones,
        2000, #n_pob
        1000, #iteraciones
        mc.cores = 10
        )
  ) -> solucion
toc()

## 2567.239 sec elapsed
```

En el código implementado es posible leer las siguientes acciones:

- Fraccionamiento de la tabla de ejecutivos en tablas regionales.
- Fraccionamiento de la tabla de gerentes en tablas regionales.

- Unión de las tablas regionales de ejecutivos y gerentes.
- Configuración de las tablas regionales por parejas, cada pareja de tablas contiene una tabla de ejecutivos y otra de gerentes. Existe una pareja de tablas por cada región.
- Ejecución dentro del bucle paralelizado, para cada región, de las siguientes tareas:
  - Cálculo de asignación óptima entre los ejecutivos y los gerentes de la misma región.
  - Uso de 1000 iteraciones en este cálculo.
  - Uso de generaciones de 2000 individuos en este cálculo.
- Guardar el resultado en un objeto en la memoria.

Guardado

La solución generada se evalúa y se guarda en los archivos correspondientes. El proceso de evaluación de la solución coincide parcialmente con el presentado anteriormente. No obstante resulta importante hacer explícitos algunos resultados con el fin de tomar decisiones al respecto.

```
solucion %$%
  map(optimos, extract2, "soluciones") %>%
  lapply(extract2, 1) %>%
  bind_rows() -> tabla_asignacion

tabla_asignacion %$% sum(clientes) -> cli_tot

tabla_asignacion %$% sum(marca_a) -> cli_a

tabla_asignacion %$% sum(marca_b) -> cli_b

tabla_asignacion %>%
  group_by(cod_gte_inv) %>%
  summarise(
    tiempo_restante = mean(tiempo_restante),
    marca_a = sum(marca_a),
    marca_b = sum(marca_b),
    clientes = sum(clientes),
    tiempo_usado = sum(tiempo_x_ejecutivo)
  ) -> tabla_gerentes

tabla_gerentes %>%
  filter(tiempo_restante > tiempo_usado) %$%
  list(
    a = sum(marca_a),
```

```
    b = sum(marca_b),
    r  = cli_tot - sum(clientes),
    g = nrow(.),
    f = (nrow(.) + 1  == nrow(tabla_gerentes))
    ) -> asignados

asignados

## $a
## [1] 4617
##
## $b
## [1] 11099
##
## $r
## [1] 4415
##
## $g
## [1] 80
##
## $f
## [1] TRUE

with(asignados,  - (a + b)/r - g - 100*f)

## [1] -183.5597

with(asignados, (a + b)/r) -> obj_fun
obj_fun

## [1] 3.559683

inner_join(
  select(
    pcac_mac_gpi_clientes,
    num_doc_cli,
    cod_tipo_doc_cli,
    cod_ejec_bco,
  ),
  select(
    tabla_asignacion,
    cod_ejec_bco,
    cod_gte_inv
  )
) %>%
  inner_join(
    select(
      tabla_gerente_tiempo_region,
      -cod_region_gte_inv,
      -tiempo_restante
    )
  ) -> resultado
```

```
names(resultado)

## [1] "num_doc_cli"      "cod_tipo_doc_cli" "cod_ejec_bco"     "cod_gte_inv"
## [5] "num_doc_gte_inv"

write_rds(resultado, "01_datos/asignacion.rds")
write_csv(resultado, "01_datos/asignacion.csv")
```

El código presentado contiene las siguientes tareas.

- De la tabla generada extraer las tablas de soluciones, una para cada región.
- De cada tabla de soluciones extraer la asignación con mejor calificación, una para cada región.
- Unir estas asignaciones conformando una asignación a nivel nacional.
- Calcular los clientes totales.
- Agrupar la asignación por gerente.
- Calcular:
  - El tiempo restante por gerente.
  - La cantidad de clientes de tipo A.
  - La cantidad de clientes de tipo B
  - El total del tiempo usado.
  - La cantidad de tiempo requerido.
- Guardando estos cálculos en una tabla nueva.
- Filtra los gerentes cuyo tiempo requerido es menor que el tiempo disponible. Con esto se garantiza la evaluación sobre la parte factible de la asignación.
- Realiza los cálculos de totales para los clientes tipo A (a), tipo B (b), y los restantes (r). Adicionalmente calcula el número de gerentes asignados (g) y si la asignación completa es factible (f).
- Muestra los resultados obtenidos de manera individual.
- Evalúa ambas funciones objetivo.
- Retira los ejecutivos de banco no asignados.
- Escribe los datos de la manera solicitada en formato rds y csv.

Conclusión

El objetivo de la prueba fue cumplido satisfactoriamente.

Luego de la implementación de un algoritmo evolutivo se obtiene una solución con las siguientes características.

- **Total de clientes tipo A asignados:** 4617
- **Total de clientes tipo A por asignar:** 878
- **Total de clientes tipo B asignados:** 11099
- **Total de clientes tipo B por asignar:** 1628
- **Total de clientes asignados:** 29730
- **Total de clientes por asignar:** 4415
- **Puntuación de la función objetivo:** 3.5596829

Con esto finaliza el ejercicio.