

Actividad Evaluable 1

2026-01-19

Análisis de logs de servidor usando R (parte II)

Aquí encontramos las librerías usadas para la realización de la Actividad.

```
#install.packages("readr")
#install.packages("dplyr")
#install.packages("tidyverse")
#install.packages("lubridate")
#install.packages("mltools")
#install.packages("data.table")
#install.packages("ggplot2")
#install.packages("stringr")
#install.packages("summarytools")

library("readr")
library("dplyr")

##
## S'està adjuntant el paquet: 'dplyr'

## Els següents objectes estan emmascarats des de 'package:stats':
##   filter, lag

## Els següents objectes estan emmascarats des de 'package:base':
##   intersect, setdiff, setequal, union

library("tidyverse")
library("lubridate")

##
## S'està adjuntant el paquet: 'lubridate'

## Els següents objectes estan emmascarats des de 'package:base':
##   date, intersect, setdiff, union

library("mltools")
```

```

## 
## S'està adjuntant el paquet: 'mltools'

## L'objecte següent està emmascatat per 'package:tidyR':
## 
##     replace_na

library("data.table")

## 
## S'està adjuntant el paquet: 'data.table'

## Els següents objectes estan emmascarats des de 'package:lubridate':
## 
##     hour, isoweek, isoyear, mday, minute, month, quarter, second, wday,
##     week, yday, year

## Els següents objectes estan emmascarats des de 'package:dplyr':
## 
##     between, first, last

library("ggplot2")
library("stringr")
library("summarytools")

```

Obtención y carga de los Datos:

Descomprimir el fichero comprimido que contiene los registros del servidor y, a partir de los datos extraídos, cargar en data frame los registros con las peticiones servidas. Empezamos por descomprimir el archivo donde se encuentran nuestros datos.

```

zip <- "./epa-http.zip"
unzip(zipfile = zip)

```

Como estos son logs, podemos usar la función *read_log()* para almacenarlos en formato dataframe (más adelante se explicará el proceso de limpieza que hemos realizado).

```

columnas <- c("IP", "Timestamp", "Peticion", "CodigoRespuesta", "BytesReply")
df <- read_log("epa-http.csv", col_names = columnas) %>%
  mutate(BytesReply = replace_na(BytesReply, 0)) %>%
  mutate(Timestamp = as.POSIXct(strptime(Timestamp, format = "%d:%H:%M:%S"))) %>%
  mutate(
    Metodo = as.factor(str_split(Peticion, " ", simplify = TRUE)[,1]),
    URL = as.factor(str_split(Peticion, " ", simplify = TRUE)[,2]),
    Version_proto = as.factor(str_split(Peticion, " ", simplify = TRUE)[,3]),
  )

## 
## -- Column specification ----- 
## cols(

```

```

##   IP = col_character(),
##   Timestamp = col_character(),
##   Peticion = col_character(),
##   CodigoRespuesta = col_double(),
##   BytesReply = col_double()
## )

## Warning: 21 parsing failures.
##   row col  expected      actual           file
## 7527 -- 5 columns 4 columns 'epa-http.csv'
## 7528 -- 5 columns 4 columns 'epa-http.csv'
## 7529 -- 5 columns 4 columns 'epa-http.csv'
## 7549 -- 5 columns 4 columns 'epa-http.csv'
## 7550 -- 5 columns 4 columns 'epa-http.csv'
## ....
## See problems(...) for more details.

head(df, 10)

## # A tibble: 10 x 8
##   IP       Timestamp     Peticion CodigoRespuesta BytesReply Metodo URL
##   <chr>    <dttm>        <chr>          <dbl>        <dbl> <fct> <fct>
## 1 141.243~ 2026-01-29 23:53:25 GET /So~          200     1497 GET  /Sof~
## 2 query2.~ 2026-01-29 23:53:36 GET /Co~          200     1325 GET  /Con~
## 3 tanuki.~ 2026-01-29 23:53:53 GET /Ne~          200     1014 GET  /New~
## 4 wpbf12-- 2026-01-29 23:54:15 GET / H~          200     4889 GET  /
## 5 wpbf12-- 2026-01-29 23:54:16 GET /ic~          200     2624 GET  /ico~
## 6 wpbf12-- 2026-01-29 23:54:18 GET /lo~          200     935  GET  /log~
## 7 140.112~ 2026-01-29 23:54:19 GET /lo~          200     2788 GET  /log~
## 8 wpbf12-- 2026-01-29 23:54:19 GET /lo~          200     124  GET  /log~
## 9 wpbf12-- 2026-01-29 23:54:19 GET /ic~          200     156  GET  /ico~
## 10 wpbf12-- 2026-01-29 23:54:19 GET /lo~         200     2788 GET  /log~
## # i 1 more variable: Version_proto <fct>

```

Incluid en el documento un apartado con la descripción de los datos analizados: fuente, tipología, descripción de la información contenida (los diferentes campos) y sus valores. Los logs que estamos analizando se tratan de logs de un servidor que venían comprimidos en un fichero .zip; estos los leemos nosotros gracias a que su almacenamiento ha sido realizado mediante un fichero .csv.

Para tener una primera visión de los datos que hemos cargado y con los que vamos a trabajar, podemos usar la función *glimpse()*.

```

glimpse(df)

## #> #> Rows: 47,748
## #> #> Columns: 8
## #> #> $ IP              <chr> "141.243.1.172", "query2.lycos.cs.cmu.edu", "tanuki.tw~"
## #> #> $ Timestamp        <dttm> 2026-01-29 23:53:25, 2026-01-29 23:53:36, 2026-01-29 ~
## #> #> $ Peticion         <chr> "GET /Software.html HTTP/1.0", "GET /Consumer.html HTT~
## #> #> $ CodigoRespuesta <dbl> 200, 200, 200, 200, 200, 200, 200, 200, 302, ~
## #> #> $ BytesReply       <dbl> 1497, 1325, 1014, 4889, 2624, 935, 2788, 124, 156, 278~
## #> #> $ Metodo            <fct> GET, GET, GET, GET, GET, GET, GET, GET, GET, ~
## #> #> $ URL              <fct> "/Software.html", "/Consumer.html", "/News.html", "/", ~
## #> #> $ Version_proto     <fct> HTTP/1.0, HTTP/1.0, HTTP/1.0, HTTP/1.0, HTTP~
```

Esto nos muestra la siguiente información de nuestro *dataframe*.

- En total nuestro *dataframe* cuenta con 47,748 filas.
- Contamos con 5 columnas distintas con el siguiente contenido y tipo de dato:
 - **IP (character)**: Se trata de la IP que está realizando la petición (no tiene por qué ser siempre una IP, también puede ser el nombre del “Usuario” que quiere acceder a dicho recurso).
 - **Timestamp (POSIXct)**: Es la fecha de cuándo llegó la petición al servidor.
 - **Peticion (character)**: Se trata de la petición que se está realizando al servidor.
 - **CodigoRespuesta (double)**: Código que indica el estado de la petición.
 - **BytesReply (double)**: Número de bytes enviados por el servidor para servir la petición.
 - **Metodo (Factor)**: Parte de la petición que indica su tipo (Hay 3 posibles opciones).
 - **URL (Factor)**: Se trata de la URL del recurso al cual se está accediendo (Hay 6561 distintos).
 - **Version_proto (Factor)**: Versión del protocolo (Solo hay dos tipos).

Limpieza de los Datos

Aprovechando que los datos a analizar son los mismos de la primera práctica, para esta entrega es imprescindible que los datos estén en formato de “datos elegantes”. Para tener los datos limpios, hemos realizado distintas correcciones respecto a cómo estos venían en crudo. - Con `read_csv()` los datos ya quedan bastante bien estructurados, y se nos crean ya las 5 primeras columnas iniciales; lo único que hemos hecho ha sido el cambio de los nombres de estas para poder identificarlas mejor. - Como tenemos filas donde encontramos valores de BytesReply nulos (cuando hay una petición con código de respuesta 404), los tratamos también para que no sean NA y los dejamos como 0, ya que al tratarse de un recurso no encontrado, el servidor no manda nada. - Luego, para las fechas de los logs, las pasamos a un formato de fecha como lo es POSIXct. Este formato nos será útil para más adelante ya que nos divide en varias partes las fechas. - Finalmente dividimos la columna *Peticion* (conservando esta en nuestro df) en tres columnas nuevas que son **Metodo**, **URL** y **Version_proto** y las almacenamos como Factores.

Exploración de Datos

Identificar el número único de usuarios que han interactuado directamente con el servidor de forma segregada según si los usuarios han tenido algún tipo de error en las distintas peticiones ofrecidas por el servidor. Para conseguir el número de usuarios únicos que han interactuado con el servidor, teniendo en cuenta si han tenido algún tipo de error, lo que hacemos es realizar una búsqueda donde inicialmente filtramos todas aquellas filas de nuestro df donde se encuentre un código de respuesta superior o igual a 400. Esto es debido a que los códigos a partir del 400 significan que ha habido errores a la hora de realizar la petición. Luego, si hacemos un `distinct` de las diferentes IPs, encontraremos el nombre de usuarios que han tenido algún tipo de fallo y, para saber el número de usuarios que no han tenido fallos, el resultado será el número de IPs totales que aparecen en nuestro df, menos el número de usuarios que sí han tenido errores.

```
usuarios_con_fallos <- df %>%
  filter(CodigoRespuesta >= 400) %>%
  distinct(IP) %>%
  nrow()

total_usuarios <- length(unique(df$IP))

usuarios_sin_fallos <- total_usuarios - usuarios_con_fallos

cat("El total de usuarios distintos es", total_usuarios, "\n")
```

```

## El total de usuarios distintos es 2333

cat("El número de usuarios que han sufrido algún tipo de fallo es",usuarios_con_fallos, "\n")

## El número de usuarios que han sufrido algún tipo de fallo es 192

cat("El número de usuarios que NO han sufrido ningún tipo de fallo es",usuarios_sin_fallos, "\n")

## El número de usuarios que NO han sufrido ningún tipo de fallo es 2141

```

Si queremos saber cuáles son estos errores, los podemos obtener filtrando de la misma manera de antes, y haciendo una obtención única de cada uno de los códigos de respuesta que acabamos de filtrar.

```

Fallos <- df %>%
  filter(CodigoRespuesta >= 400) %>%
  distinct(CodigoRespuesta)

```

Fallos

```

## # A tibble: 5 x 1
##   CodigoRespuesta
##       <dbl>
## 1         404
## 2         403
## 3         501
## 4         500
## 5         400

```

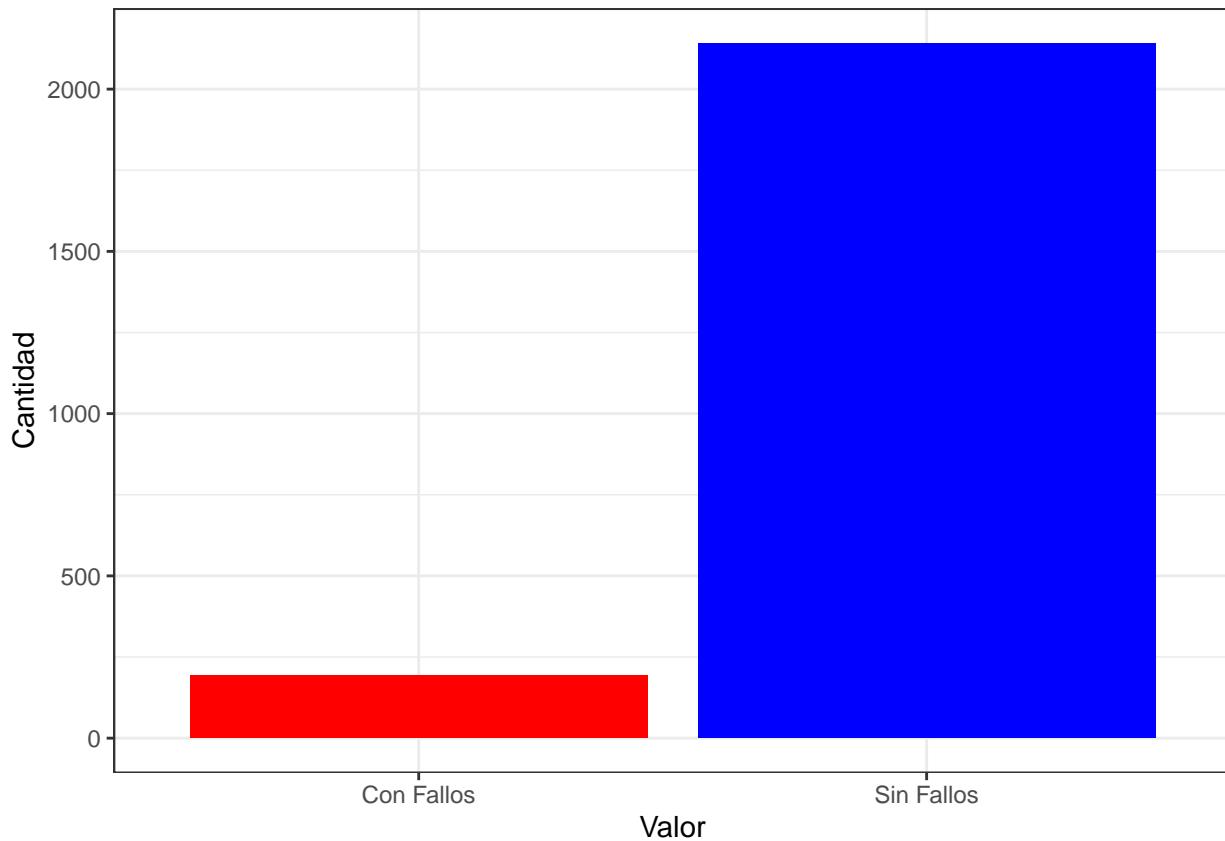
Y aquí tenemos un gráfico donde podemos ver claramente que la cantidad de usuarios que han sufrido algún tipo de error por parte del servidor es muy inferior a los usuarios que no han sufrido ninguno.

```

usuarios_fallos <- data.frame(
  Valor = c("Sin Fallos", "Con Fallos"),
  Cantidad = c(usuarios_sin_fallos, usuarios_con_fallos)
)

ggplot(usuarios_fallos, aes(x=Valor, y=Cantidad, fill=Valor)) +
  geom_col() +
  scale_fill_manual(values = c("red", "blue")) +
  theme_bw() +
  theme(legend.position="none")

```



Análisis de Datos

Analizar los distintos tipos de peticiones HTTP (GET, POST, PUT, DELETE) gestionadas por el servidor, identificando la frecuencia de cada una de estas. Repetir el análisis, esta vez filtrando previamente aquellas peticiones correspondientes a recursos ofrecidos de tipo imagen. Para obtener la frecuencia de los tipos de peticiones realizados por el servidor podemos usar la función *freq()* que ya nos almacena los datos de manera que podemos tener una visión de la frecuencia de cada una de las peticiones.

En nuestro caso, la petición que más se realiza son los GET (96.38%), seguido de los POST (3.4%) y finalmente tenemos los HEAD (0.22%).

```
tabla_freq <- freq(df$Metodo, report.nas = FALSE)
tabla_freq
```

```
## Frequencies
## df$Metodo
## Type: Factor
##
##          Freq      %   % Cum.
## ----- -----
##       GET    46020  96.38  96.38
##       HEAD     106   0.22  96.60
##      POST    1622   3.40 100.00
##    Total    47748 100.00 100.00
```

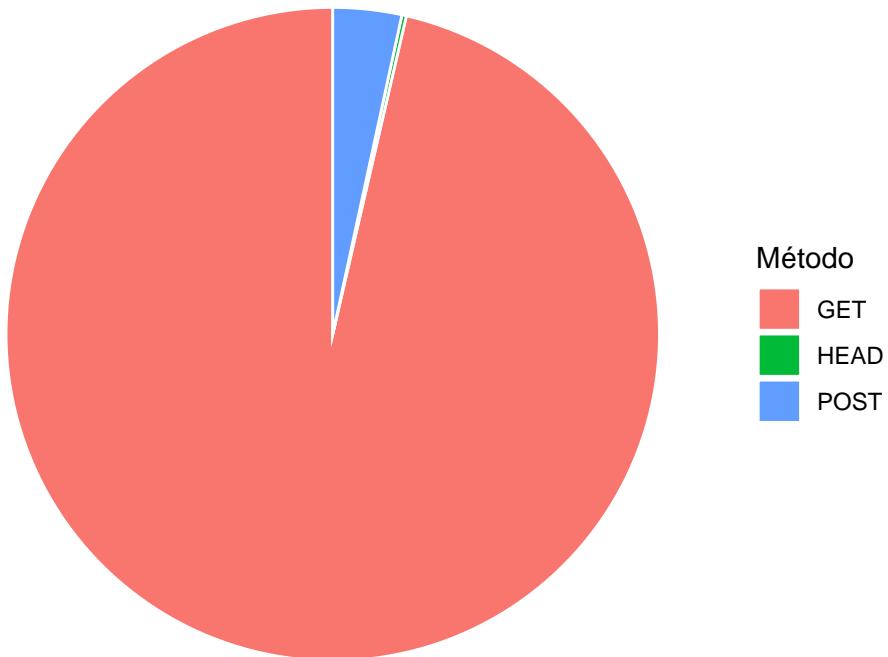
Aquí podemos observar los resultados obtenidos en formato de *Pie chart*.

```
datos_pie <- as.data.frame(tabla_freq)

datos_pie <- datos_pie %>%
  mutate(Categoría = rownames(.)) %>%
  filter(Categoría != "Total") %>%
  filter(!is.na(Categoría) & Categoría != "<NA>")

ggplot(datos_pie, aes(x = "", y = Freq, fill = Categoría)) +
  geom_bar(stat = "identity", width = 1, color = "white") +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Distribución de Métodos", fill = "Método")
```

Distribución de Métodos



Podemos hacer lo mismo pero filtrando inicialmente por todas aquellas peticiones que contengan recursos ofrecidos del tipo imagen. Los resultados en este caso varían del análisis con todos los logs. En este caso el porcentaje de GETs aumenta y el de POST disminuye, mientras que los HEADS se mantienen.

```
patron_imagenes <- "(?i)\\.\\.(gif|jpg|jpeg|png|ico|bmp|svg)$"
df_imagenes <- df %>%
  filter(str_detect(URL, patron_imagenes))
tabla_freq_img <- freq(df_imagenes$Metodo, report.nas = FALSE)
tabla_freq_img

## Frequencies
```

```

## df_imagenes$Metodo
## Type: Factor
##
##          Freq      %  % Cum.
## -----
##      GET    22189  98.83  98.83
##     HEAD      56   0.25  99.08
##     POST     206   0.92 100.00
## Total    22451 100.00 100.00

```

Aquí tenemos el respectivo gráfico.

```

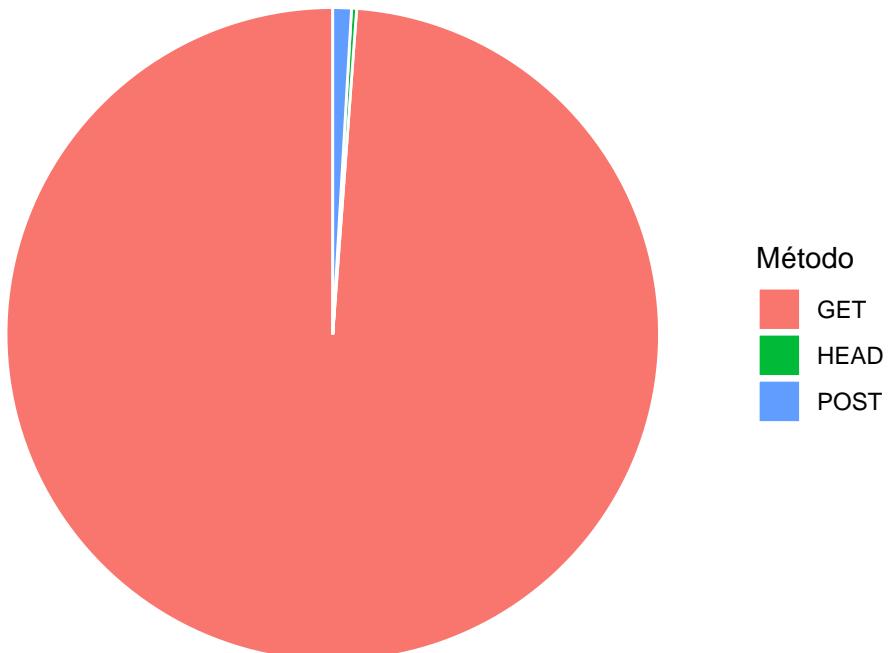
datos_pie <- as.data.frame(tabla_freq_img)

datos_pie <- datos_pie %>%
  mutate(Categoría = rownames(.)) %>%
  filter(Categoría != "Total") %>%
  filter(!is.na(Categoría) & Categoría != "<NA>")

ggplot(datos_pie, aes(x = "", y = Freq, fill = Categoría)) +
  geom_bar(stat = "identity", width = 1, color = "white") +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(title = "Distribución de Métodos", fill = "Método")

```

Distribución de Métodos

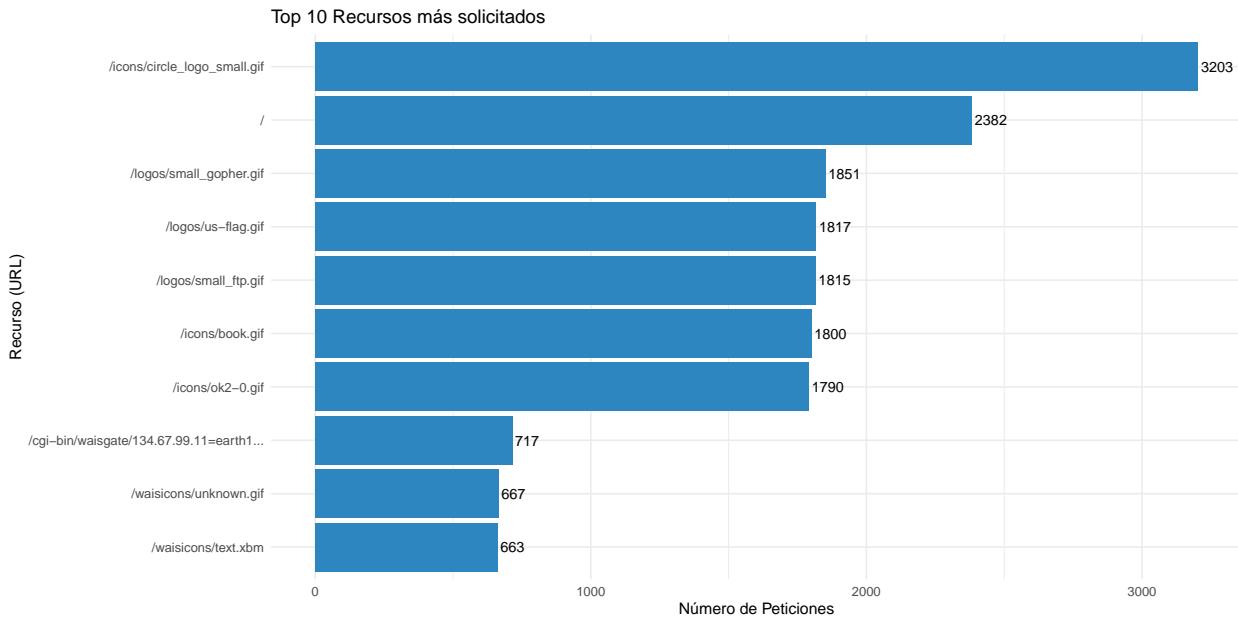


```
## Visualización de Resultados #### Generar al menos 2 gráficos distintos que permitan visualizar alguna característica relevante de los datos analizados.
```

Como gráficos a analizar, teniendo en cuenta que ya hemos mostrado algunos de estos, hemos pensado que podría ser interesante ver cuáles son los recursos que más se solicitan en nuestro servidor, y esto lo hemos logrado con un gráfico de barras ordenado que muestre la cantidad de veces que se solicitan dichos recursos. De esta manera, sabemos que el recurso más solicitado de nuestro sistema es el `/icons/circle_logo_small.gif`.

```
top_urls <- df %>%
  count(URL) %>%
  arrange(desc(n)) %>%
  slice_head(n = 10) %>%
  mutate(URL = as.character(URL))

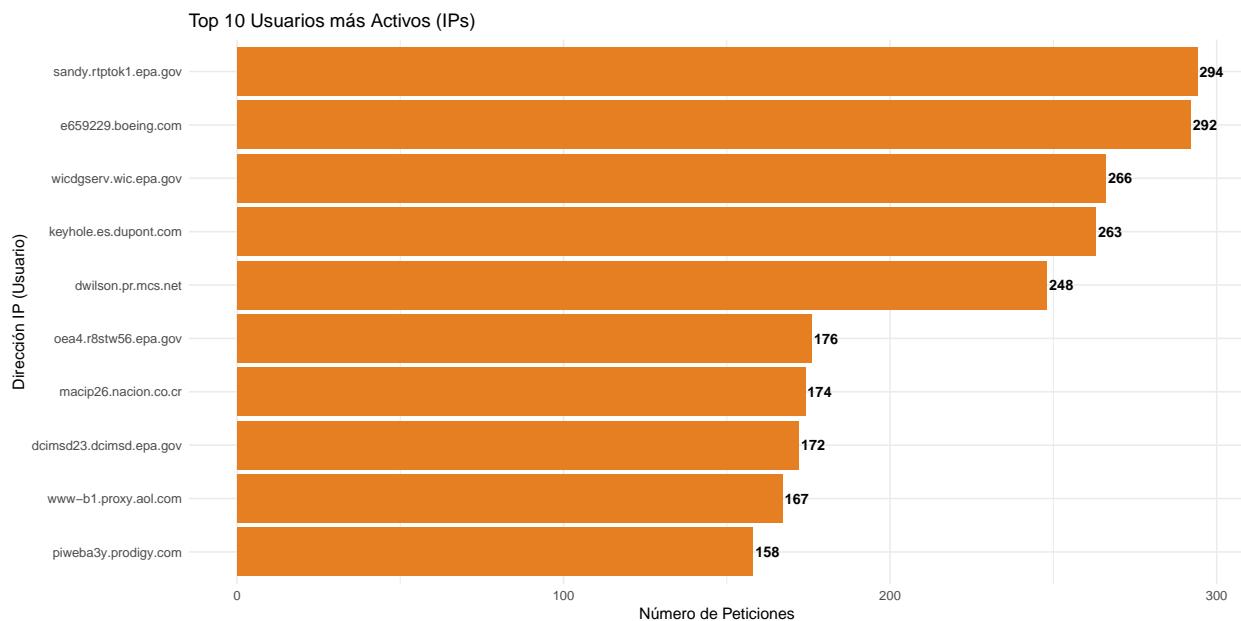
ggplot(top_urls, aes(x = reorder(str_trunc(URL, 40), n), y = n)) +
  geom_col(fill = "#2E86C1") +
  coord_flip() +
  theme_minimal() +
  labs(
    title = "Top 10 Recursos más solicitados",
    x = "Recurso (URL)",
    y = "Número de Peticiones"
  ) +
  geom_text(aes(label = n), hjust = -0.1, size = 3.5)
```



Siguiendo un poco la dinámica anterior, este gráfico muestra cuáles son los usuarios que más peticiones realizan al servidor, por lo tanto, cuáles son los usuarios más activos. En este caso el usuario más activo es `sandy.rtptok1.epa.gov`, aunque lo sigue por poco `e659229.boeing.com`.

```
top_ips <- df %>%
  count(IP) %>%
  arrange(desc(n)) %>%
  slice_head(n = 10)
```

```
# 2. Generar el Gráfico
ggplot(top_ips, aes(x = reorder(IP, n), y = n)) +
  geom_col(fill = "#E67E22") +
  coord_flip() +
  theme_minimal() +
  labs(
    title = "Top 10 Usuarios más Activos (IPs)",
    x = "Dirección IP (Usuario)",
    y = "Número de Peticiones"
  ) +
  geom_text(aes(label = n), hjust = -0.1, size = 3.5, fontface = "bold")
```



Generar un gráfico que permita visualizar el número de peticiones servidas a lo largo del tiempo. Para generar el gráfico, primero creamos una columna donde se almacenará la hora donde se produjo la petición. Luego almacenaremos el conteo de peticiones según la hora.

```
peticiones_por_tiempo <- df %>%
  mutate(Hora = floor_date(Timestamp, unit = "hour")) %>%
  count(Hora)
```

Hemos decidido generar dos gráficos distintos que muestran la misma información, pero de distinta manera. En este primero tenemos un gráfico con líneas y puntos, donde los puntos son el recuento de peticiones por hora y las líneas nos muestran cómo suben y bajan las peticiones a lo largo del día.

```
ggplot(peticiones_por_tiempo, aes(x = Hora, y = n)) +
  geom_line(color = "#0073C2", linewidth = 1) +
  geom_point(color = "#0073C2", size = 2) +
  theme_minimal() +
  labs(
    title = "Evolución del Tráfico del Servidor",
    subtitle = "Número de peticiones servidas por hora",
```

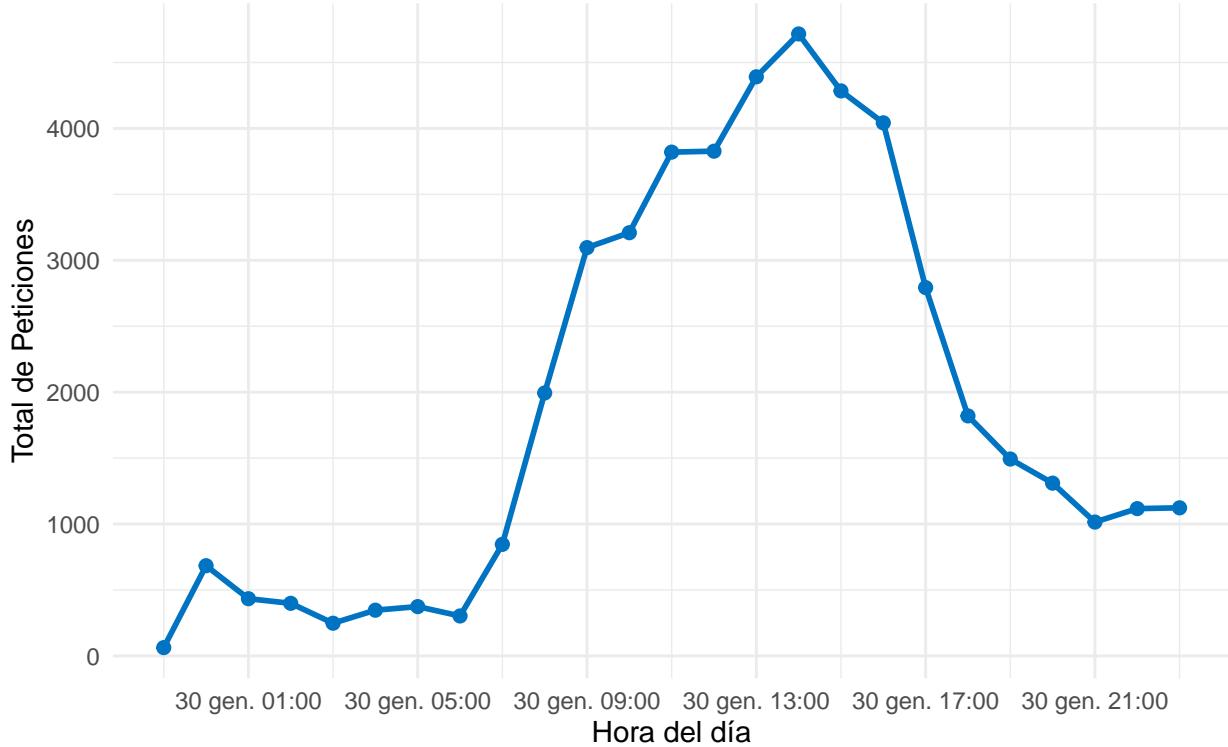
```

x = "Hora del día",
y = "Total de Peticiones"
) +
scale_x_datetime(date_labels = "%d %b %H:00", date_breaks = "4 hours")

```

Evolución del Tráfico del Servidor

Número de peticiones servidas por hora

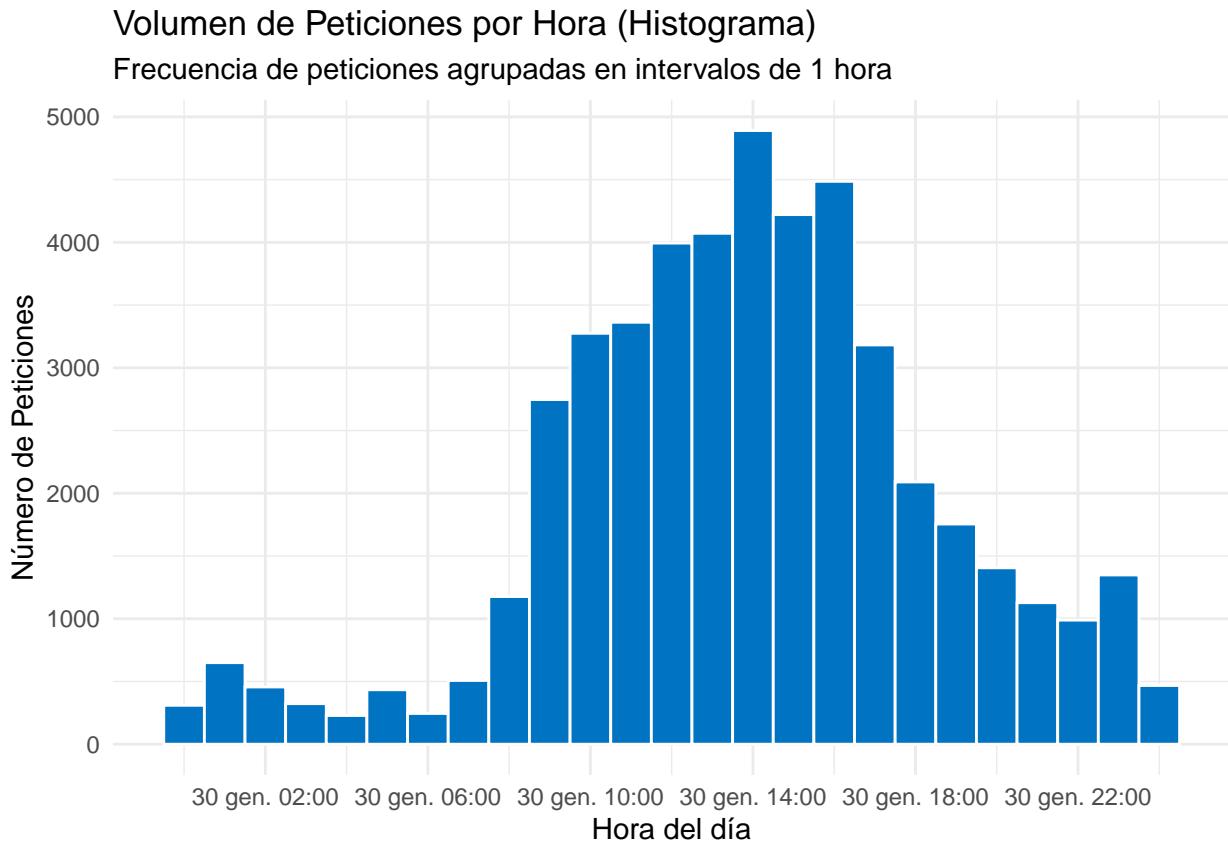


En este segundo gráfico, se muestra lo mismo, pero en forma de histograma, donde el cálculo de los valores lo hace `ggplot()`.

```

ggplot(df, aes(x = Timestamp)) +
  geom_histogram(binwidth = 3600, fill = "#0073C2", color = "white") +
  theme_minimal() +
  labs(
    title = "Volumen de Peticiones por Hora (Histograma)",
    subtitle = "Frecuencia de peticiones agrupadas en intervalos de 1 hora",
    x = "Hora del día",
    y = "Número de Peticiones"
  ) +
  scale_x_datetime(date_labels = "%d %b %H:00", date_breaks = "4 hours")

```



Clustering de datos

Utilizando un algoritmo de aprendizaje no supervisado, realizad un análisis de clustering con k-means para los datos del servidor. Para usar el K-means, necesitamos usar solamente variables numéricas, ya que dicho algoritmo usa estas variables para el cálculo de las distancias. Podemos cambiar alguna de las columnas categóricas que están almacenadas como *Factor* usando el *one hot encoding*, que crea una columna para cada valor del factor, y si a esa fila le corresponde ese valor, se marca como 1, en caso contrario 0.

```
df <- df %>%
  mutate(Sum_char_peticion = nchar(Peticion), Sum_char_ip = nchar(IP))

df$URL <- NULL

epa_http_one_hot <- one_hot(as.data.table(df), sparsifyNAs = TRUE)

epa_http_one_hot$IP <- NULL
epa_http_one_hot$Timestamp <- NULL
epa_http_one_hot$Peticion <- NULL
```

Hicimos unas cuantas ejecuciones de k-means cambiando el número de k que usábamos, y el resultado que más nos gustó fue con $k=3$.

```

#num_cluster <- 2
#result <- kmeans(epa_http_one_hot, centers = num_cluster, nstart = 25)
#result_no_na <- kmeans(na.omit(epa_http_one_hot), centers = num_cluster)

num_cluster <- 3
result <- kmeans(epa_http_one_hot, centers = num_cluster, nstart = 25)
result_no_na <- kmeans(na.omit(epa_http_one_hot), centers = num_cluster)

#num_cluster <- 4
#result <- kmeans(epa_http_one_hot, centers = num_cluster, nstart = 25)
#result_no_na <- kmeans(na.omit(epa_http_one_hot), centers = num_cluster)

```

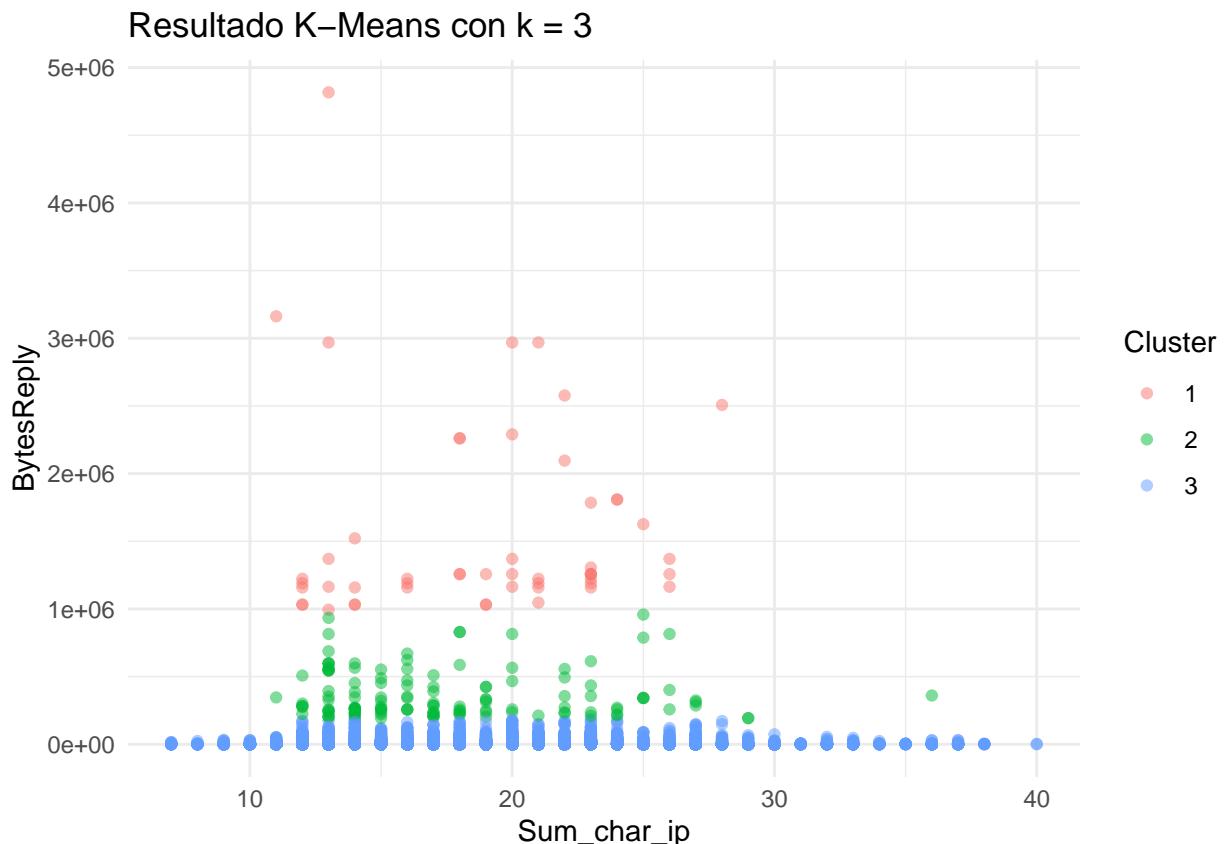
Para poder ver los distintos grupos, realizamos una gráfica donde hacemos un factor de Colors que nos permitirá ver visualmente los grupos que se han formado.

```

Colors = as.factor(result$cluster)

ggplot(epa_http_one_hot, aes(x = Sum_char_ip, y = BytesReply, color = Colors)) +
  geom_point(alpha = 0.5) +
  theme_minimal() +
  labs(
    title = paste("Resultado K-Means con k =", num_cluster),
    x = "Sum_char_ip",
    y = "BytesReply",
    color = "Cluster"
  )

```



Si analizamos los gráficos vemos que, en la mayoría de ellos, se ve muy claramente que cuando se tiene en cuenta los *BytesReply*, es cuando mejor vemos que se forman los grupos, y eso puede significar que esta variable es la más significativa por la cual se forman los grupos. Esto se debe a que la diferencia de valores distintos que hay en esta variable es mucho mayor que en las otras variables, por lo que las distancias son más significativas.

```
# Variables continuas principales (ajusta nombres según tus columnas reales)
vars_interesantes <- c("BytesReply", "Sum_char_ip", "Sum_char_peticion", "CodigoRespuesta")
todas_vars <- names(epa_http_one_hot)

# Loop anidado manual
for (var_x in vars_interesantes) {
  for (var_y in todas_vars) {

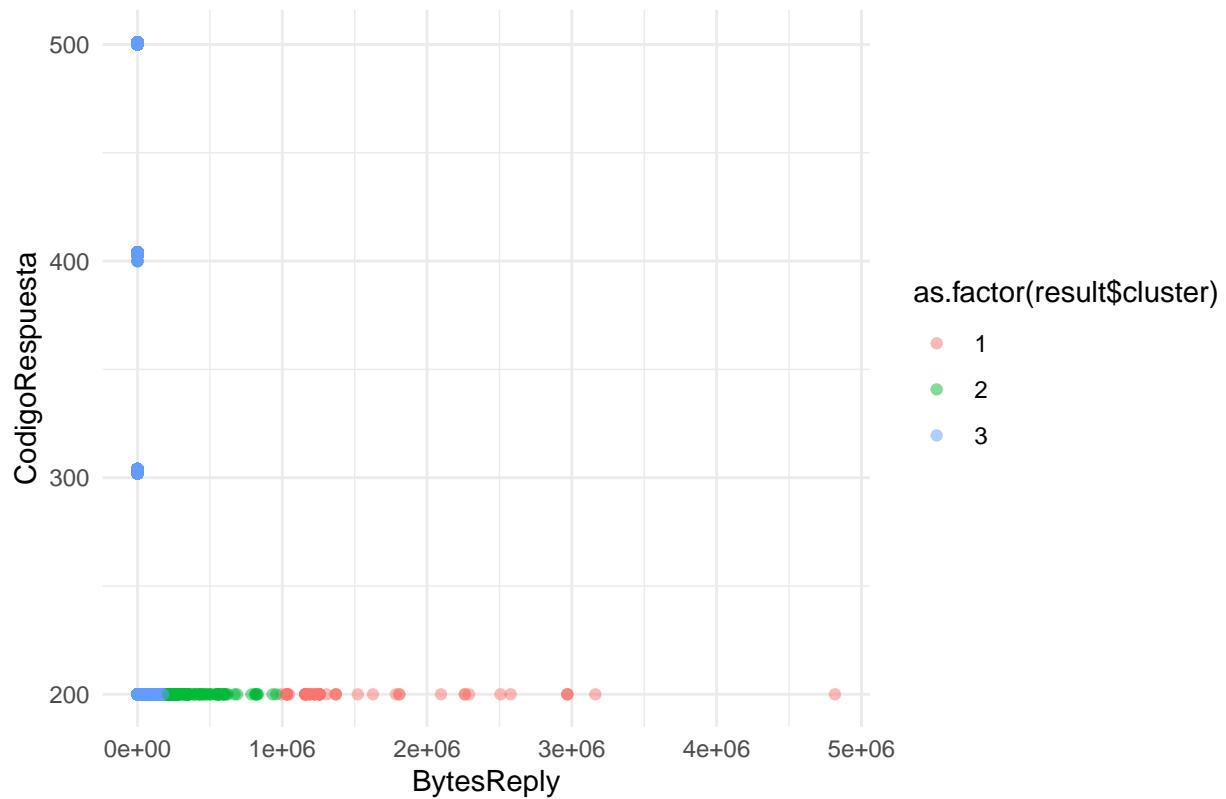
    # Evitamos comparar una variable consigo misma
    if (var_x != var_y) {

      p <- ggplot(epa_http_one_hot, aes(x = .data[[var_x]], y = .data[[var_y]], color = as.factor(result)))
      geom_point(alpha = 0.5) +
        theme_minimal() +
        labs(title = paste(var_x, "vs", var_y))

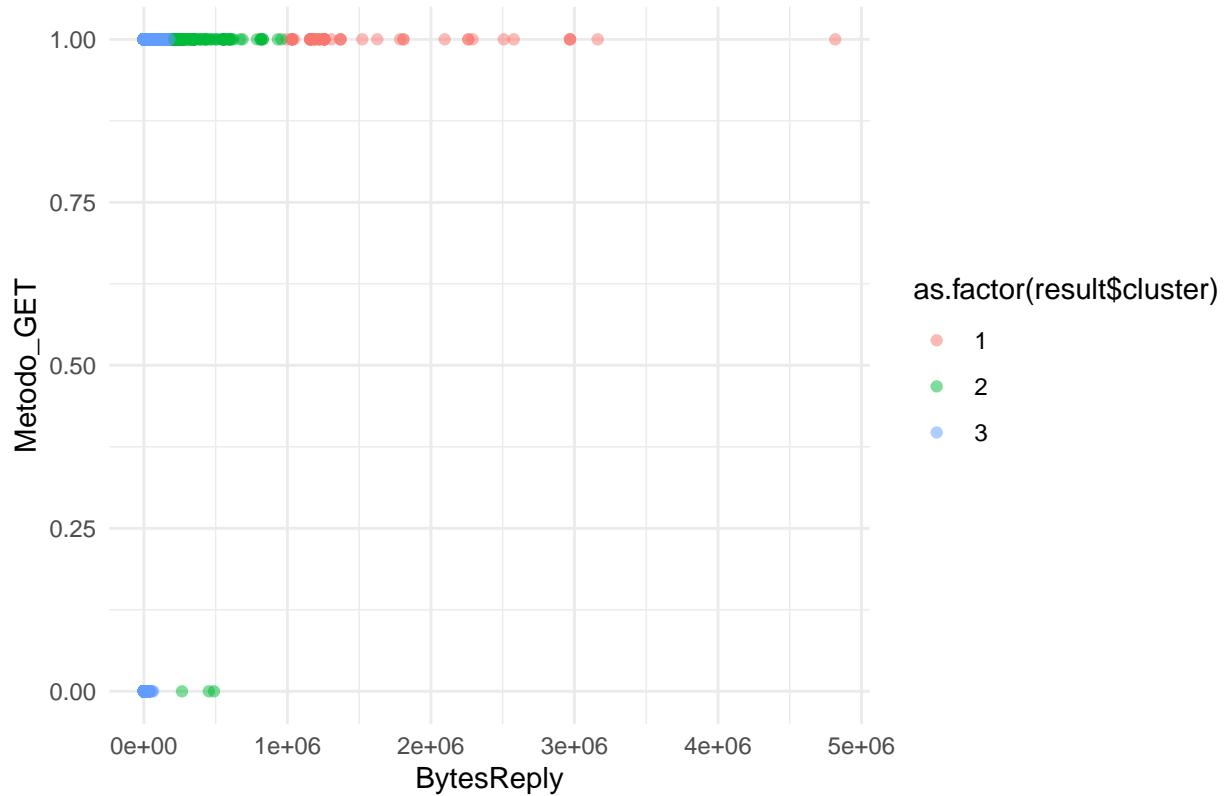
      print(p)

      # Pausa pequeña para que a R le de tiempo a renderizar si lo ves en pantalla
      Sys.sleep(0.1)
    }
  }
}
```

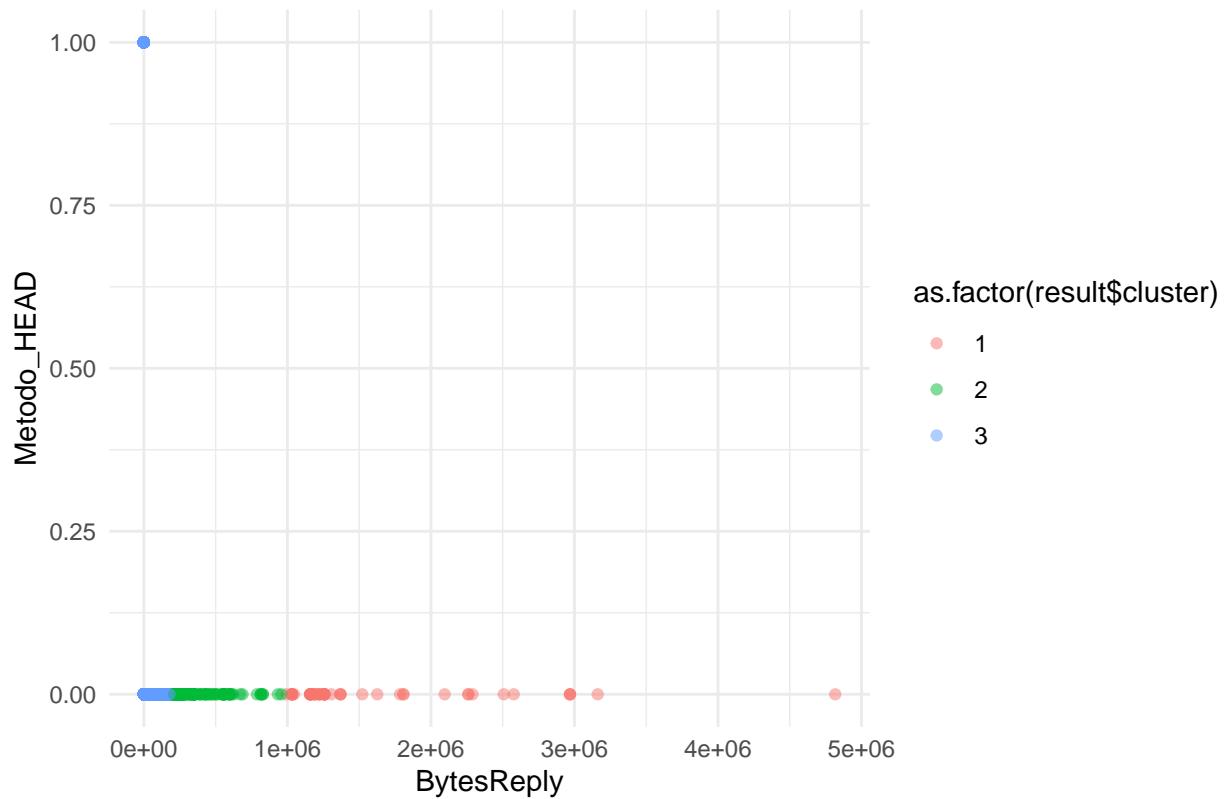
BytesReply vs CódigoRespuesta



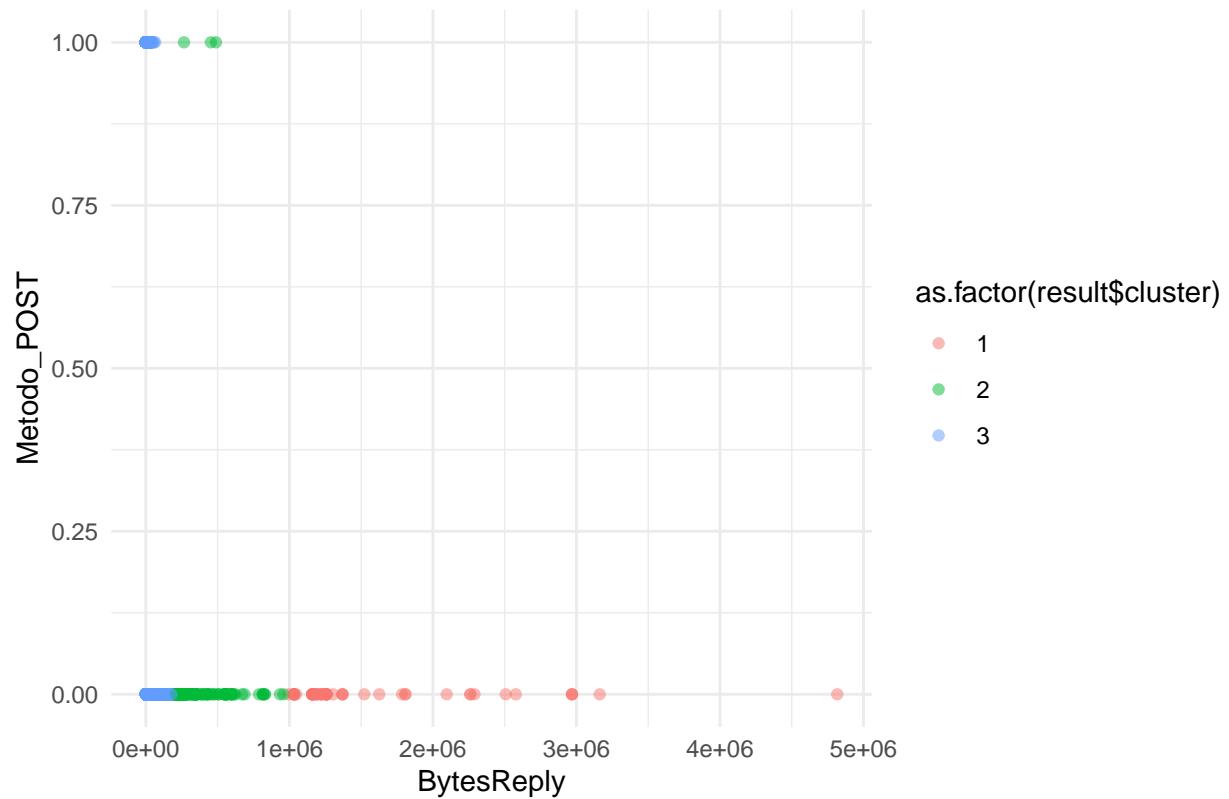
BytesReply vs Metodo_GET



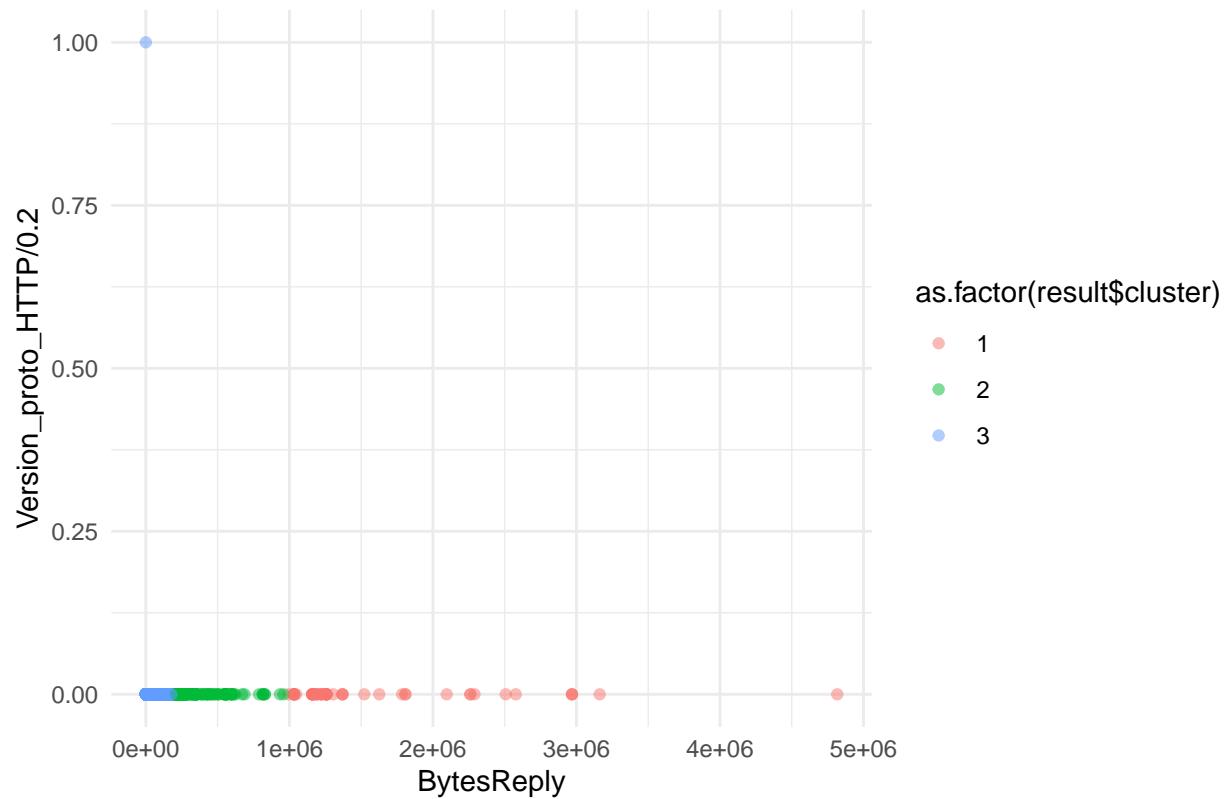
BytesReply vs Metodo_HEAD

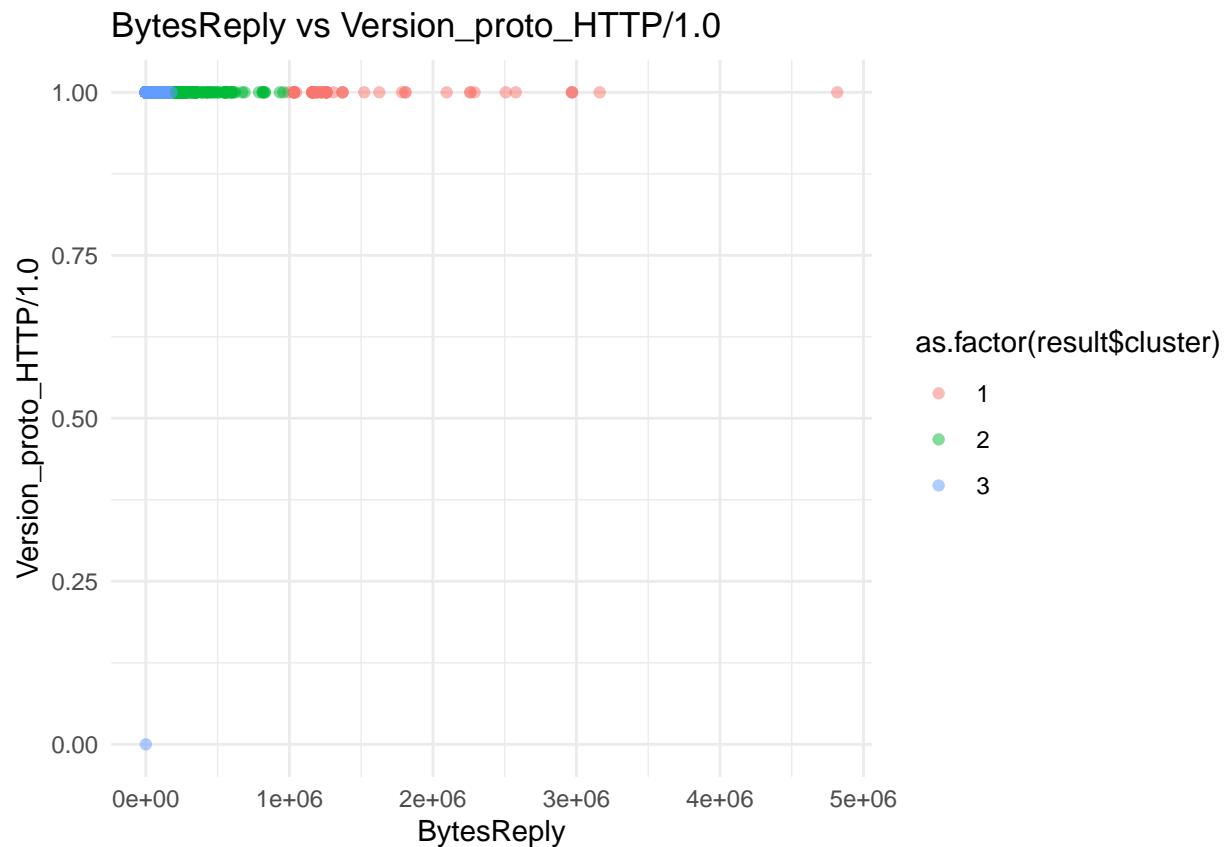


BytesReply vs Metodo_POST

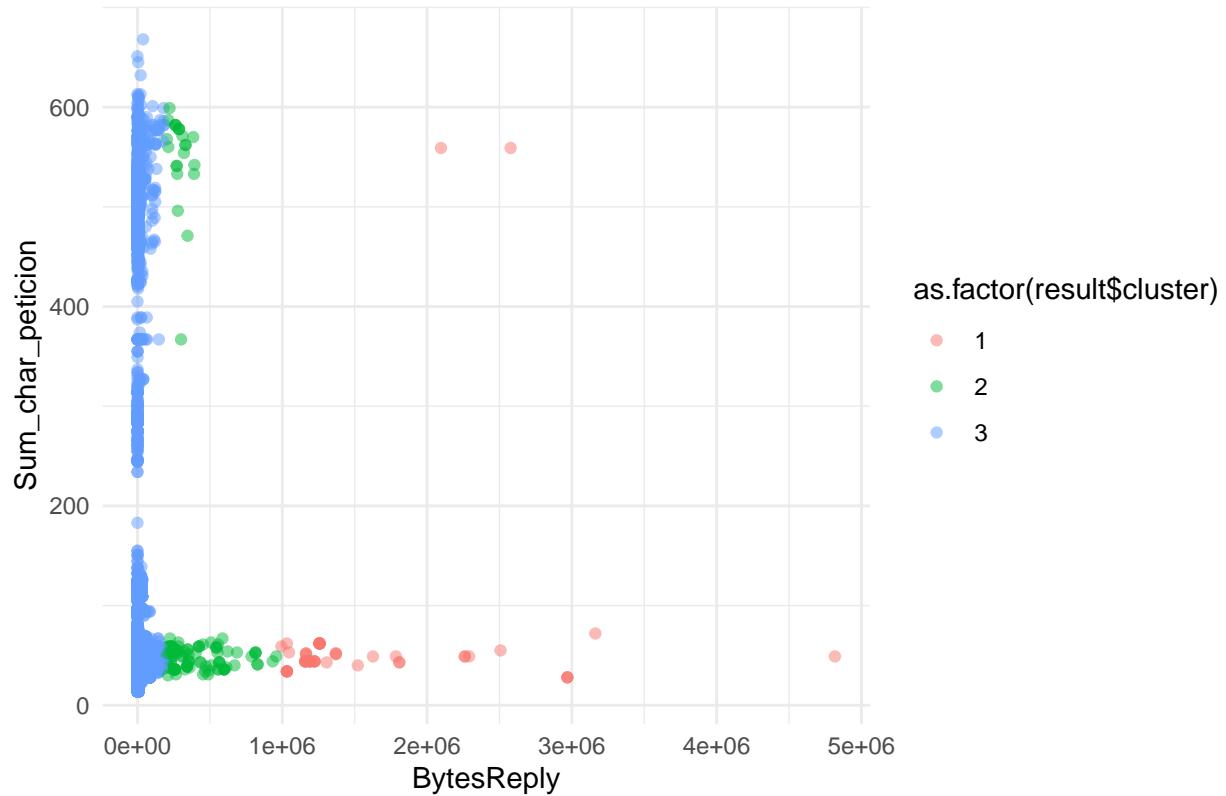


BytesReply vs Version_proto_HTTP/0.2

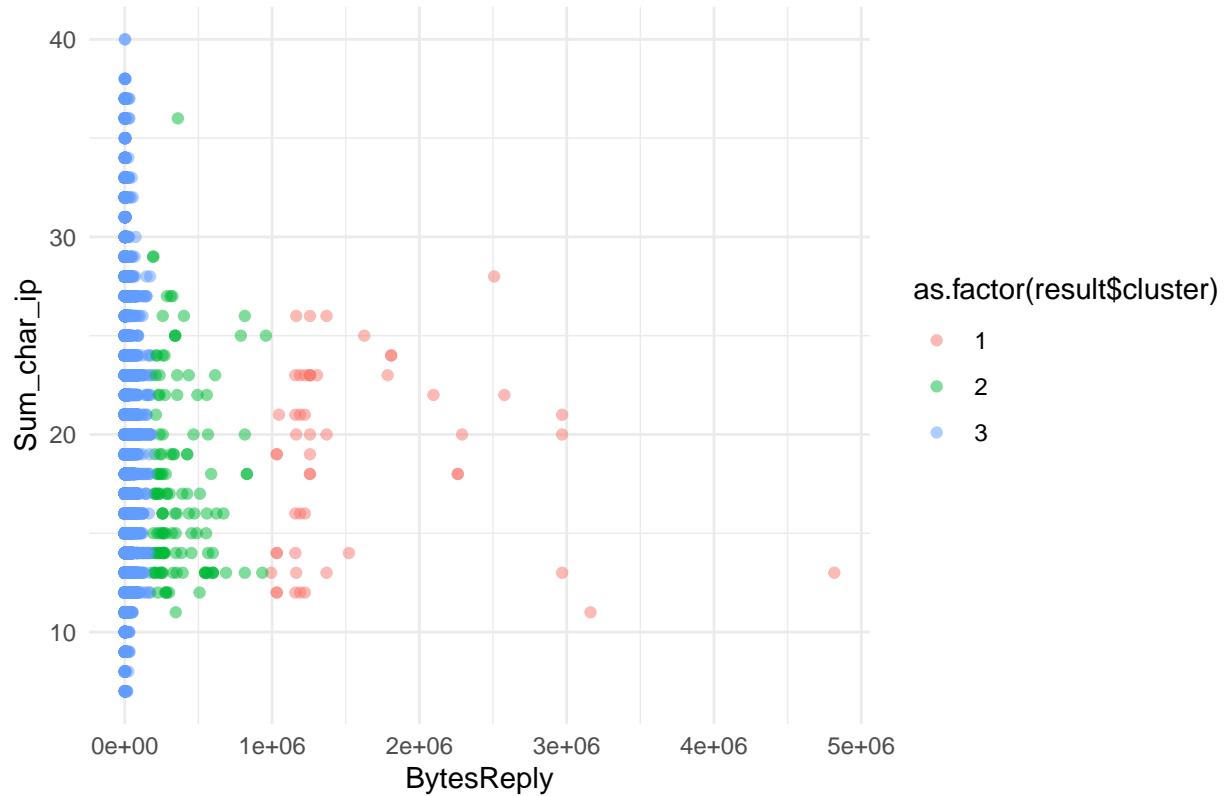




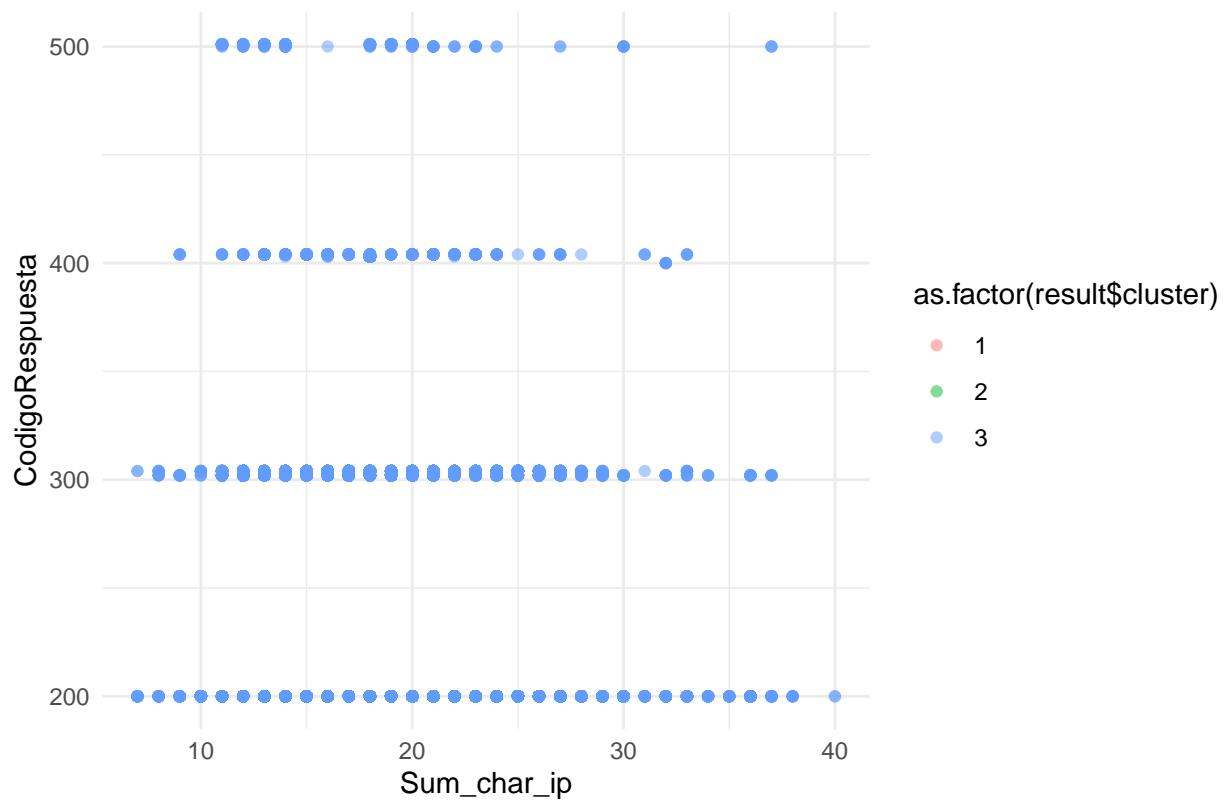
BytesReply vs Sum_char_peticion

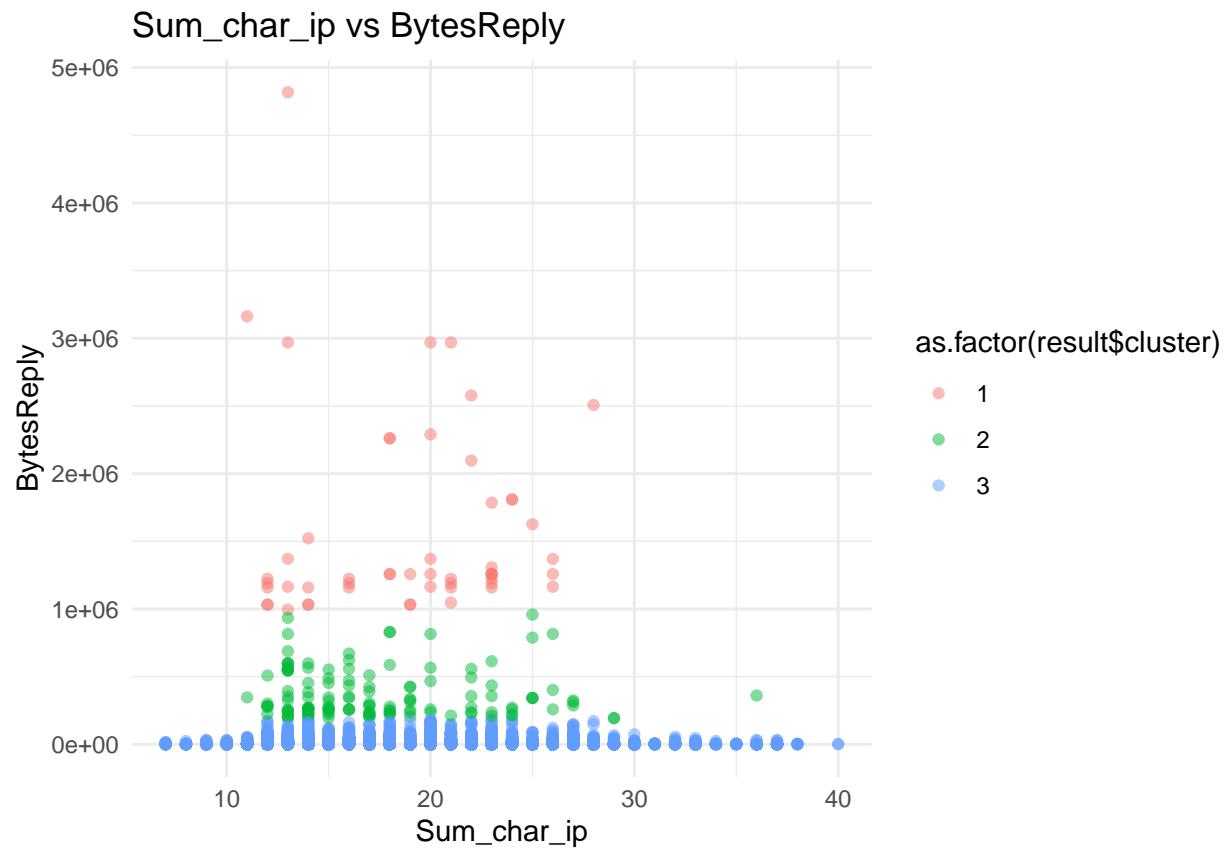


BytesReply vs Sum_char_ip

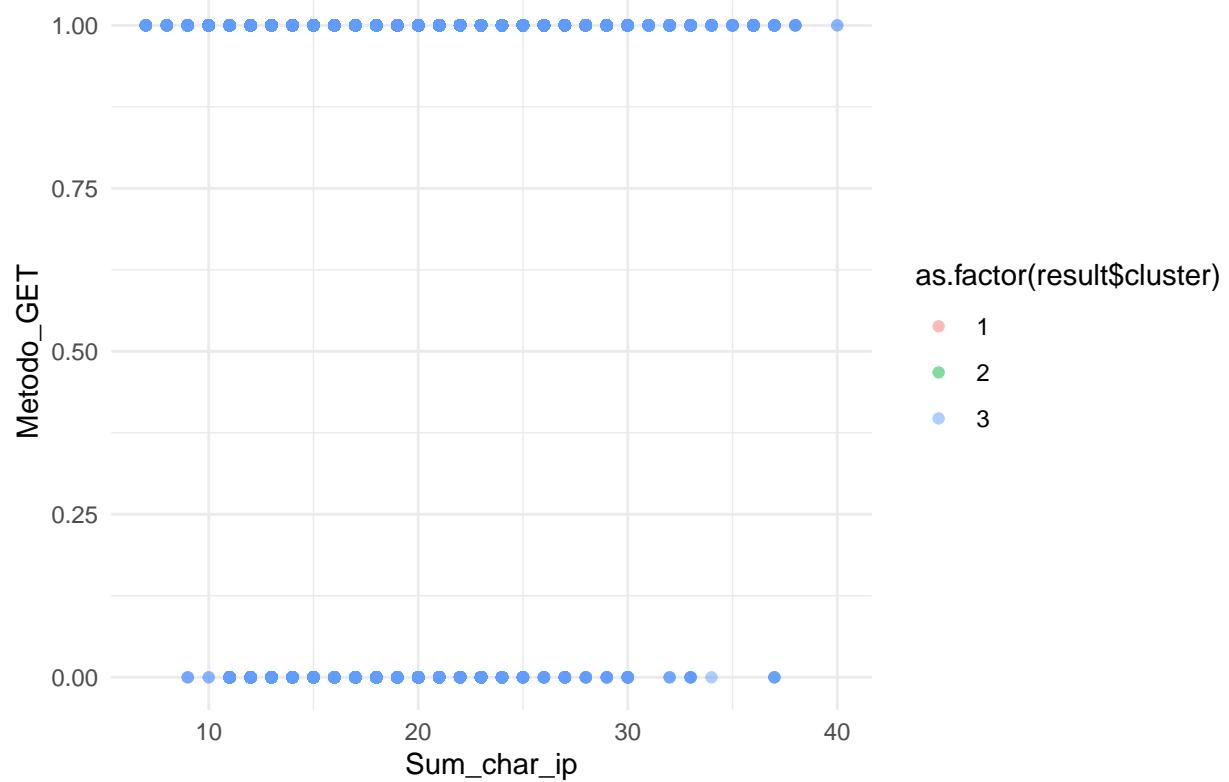


Sum_char_ip vs CódigoRespuesta

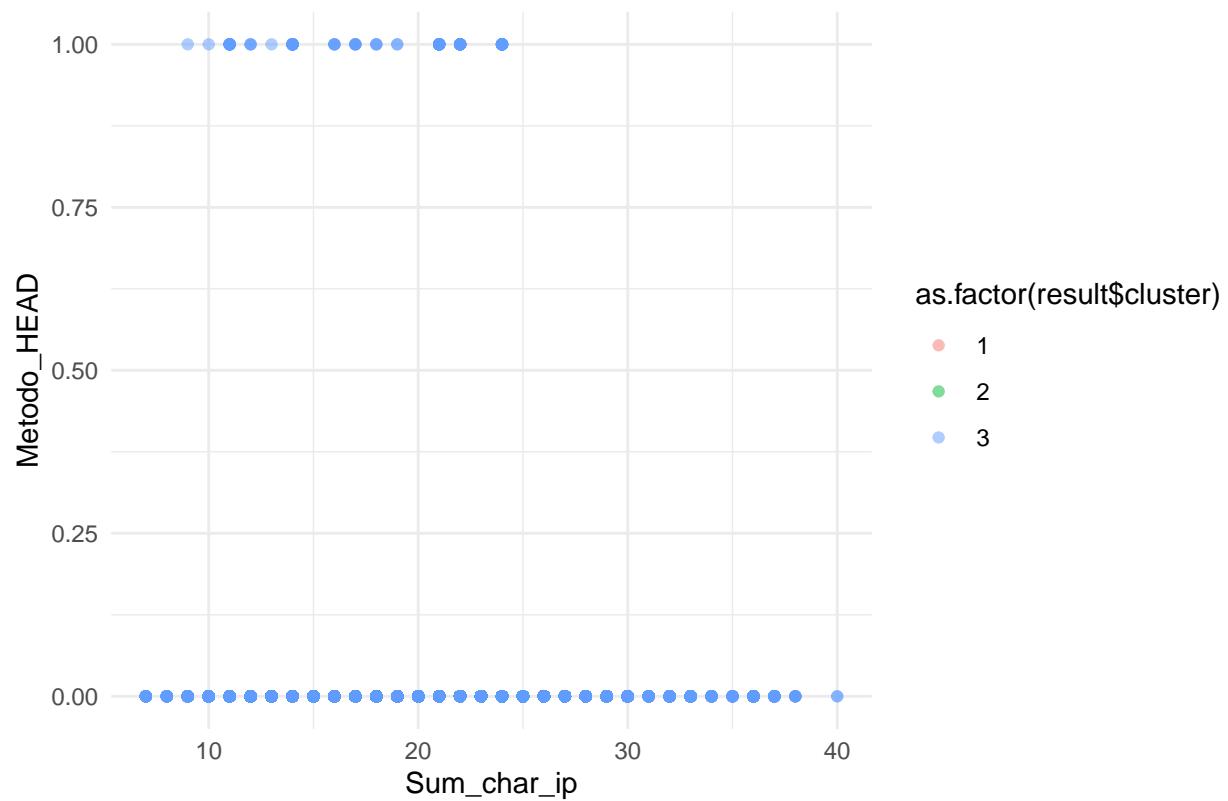




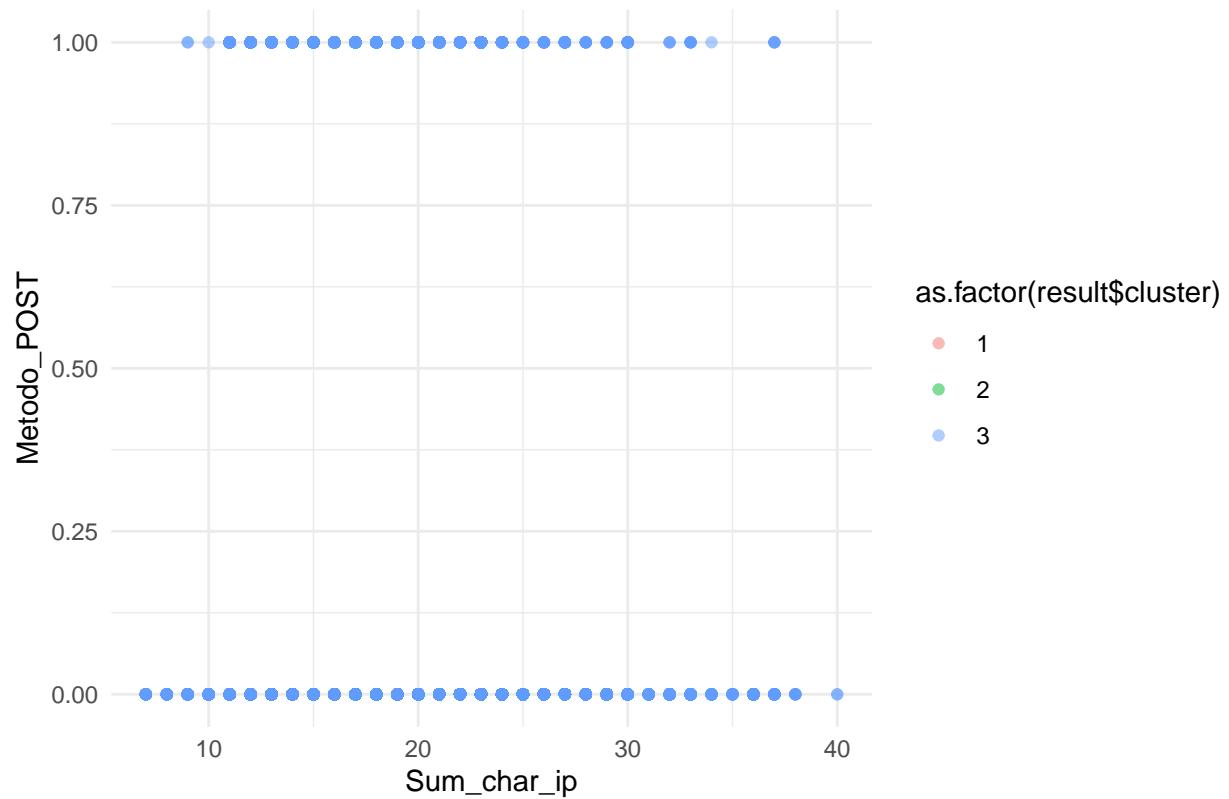
Sum_char_ip vs Metodo_GET



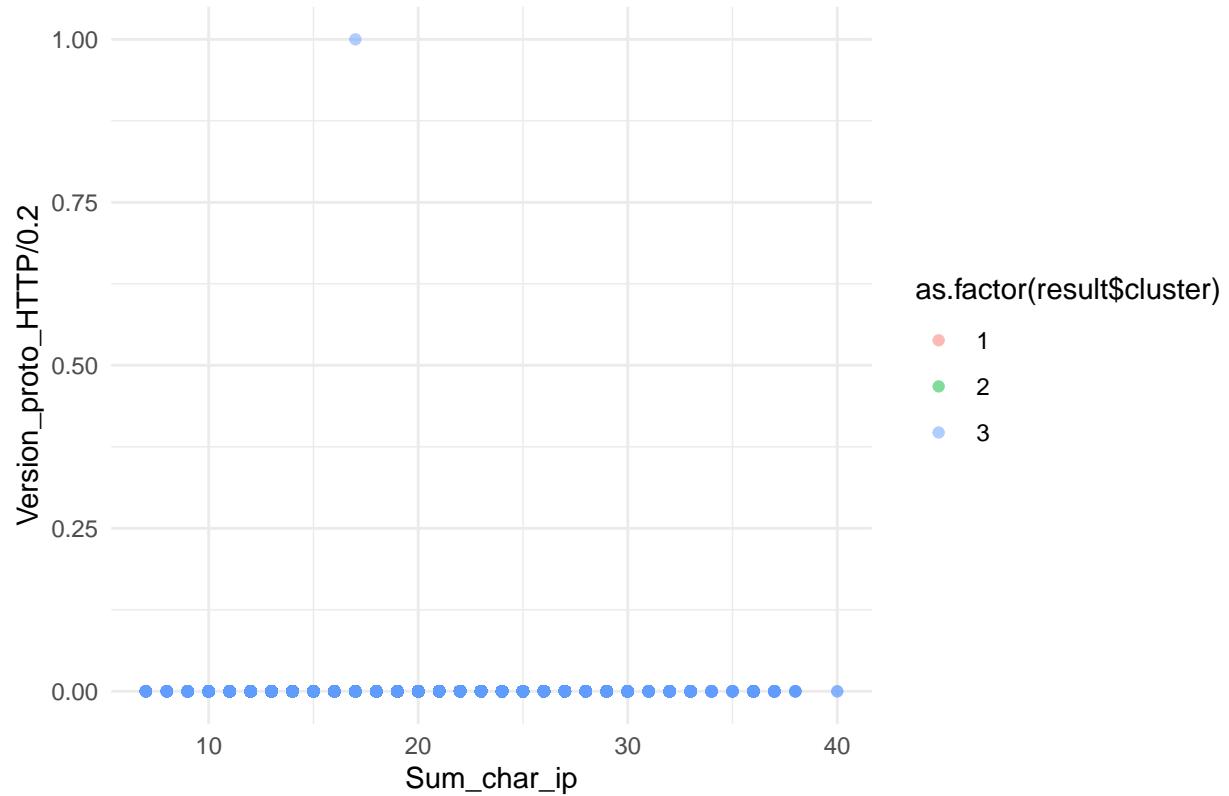
Sum_char_ip vs Metodo_HEAD



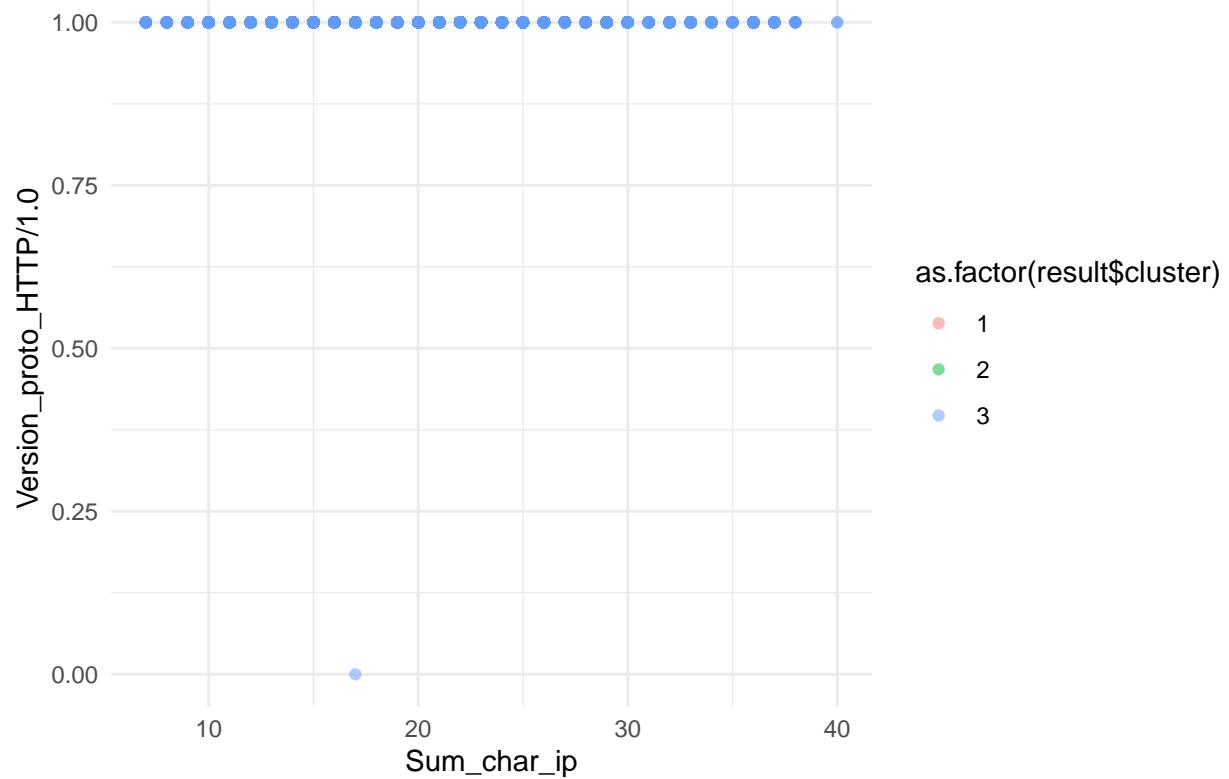
Sum_char_ip vs Metodo_POST



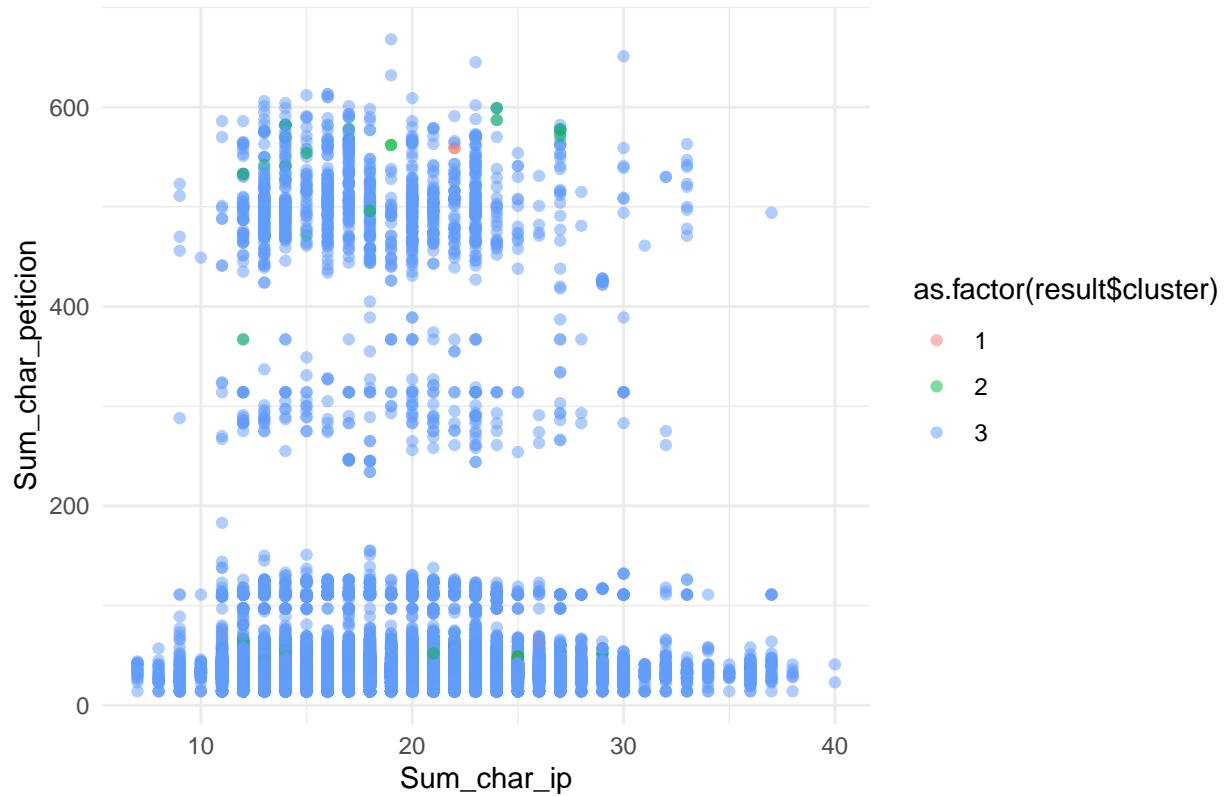
Sum_char_ip vs Version_proto_HTTP/0.2



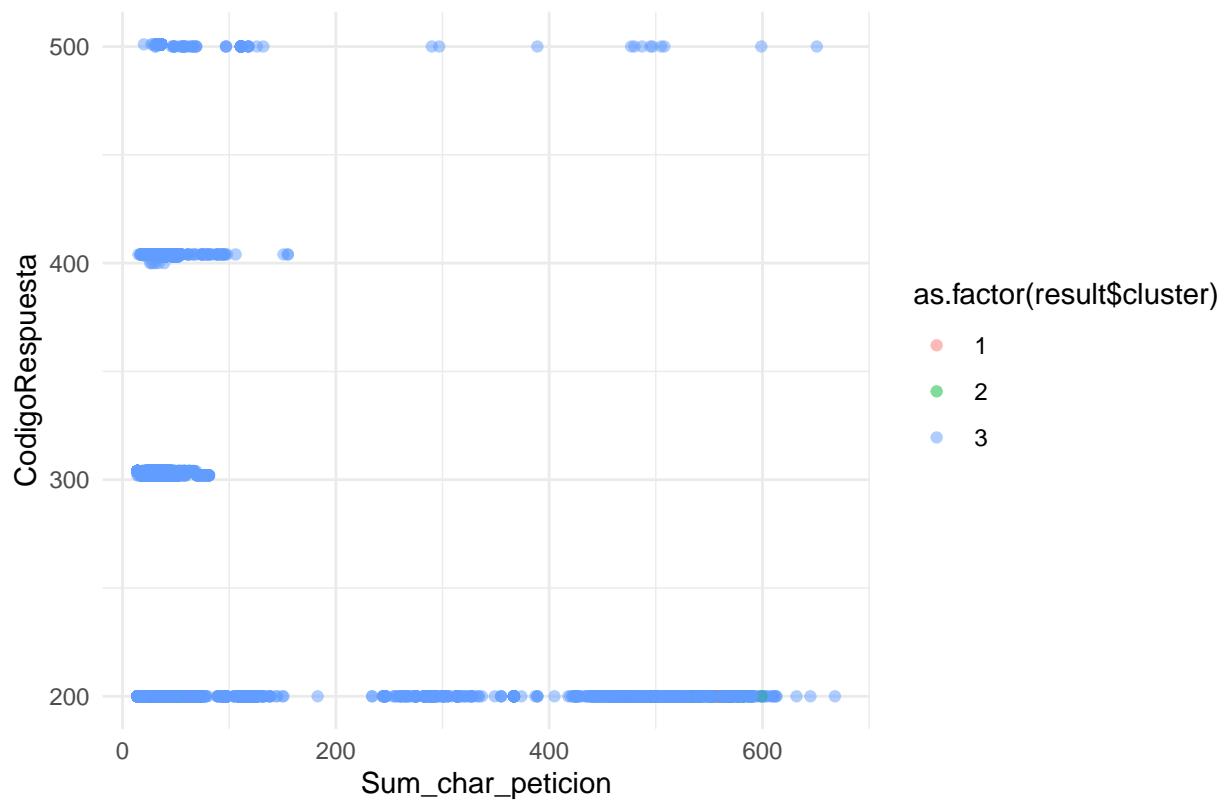
Sum_char_ip vs Version_proto_HTTP/1.0

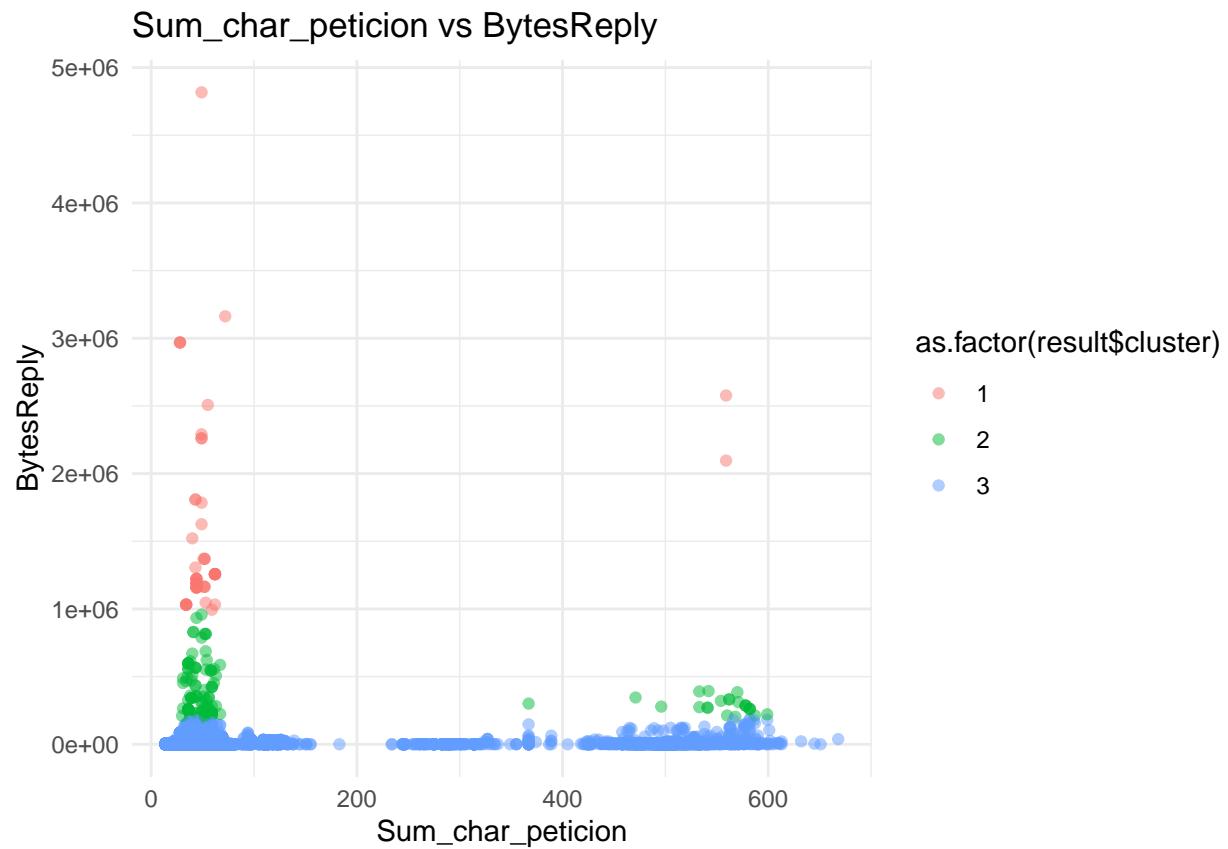


Sum_char_ip vs Sum_char_peticion

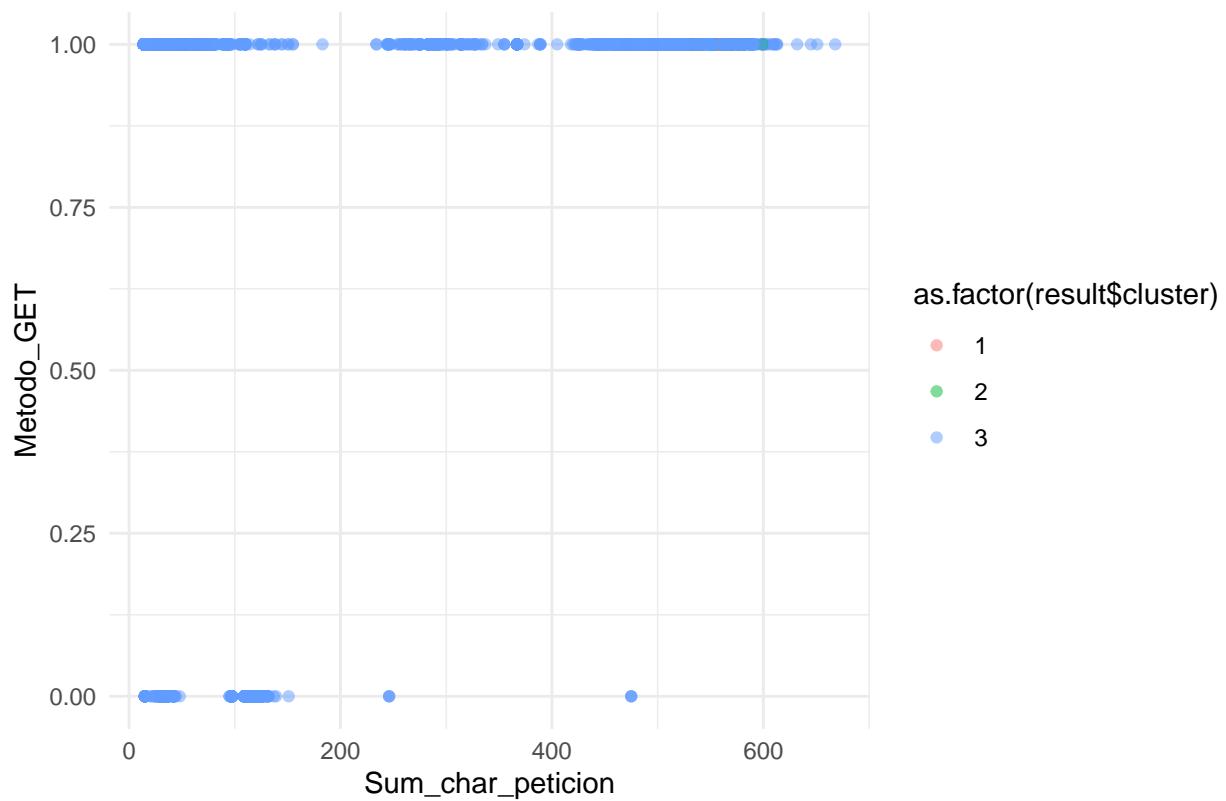


Sum_char_peticion vs CódigoRespuesta

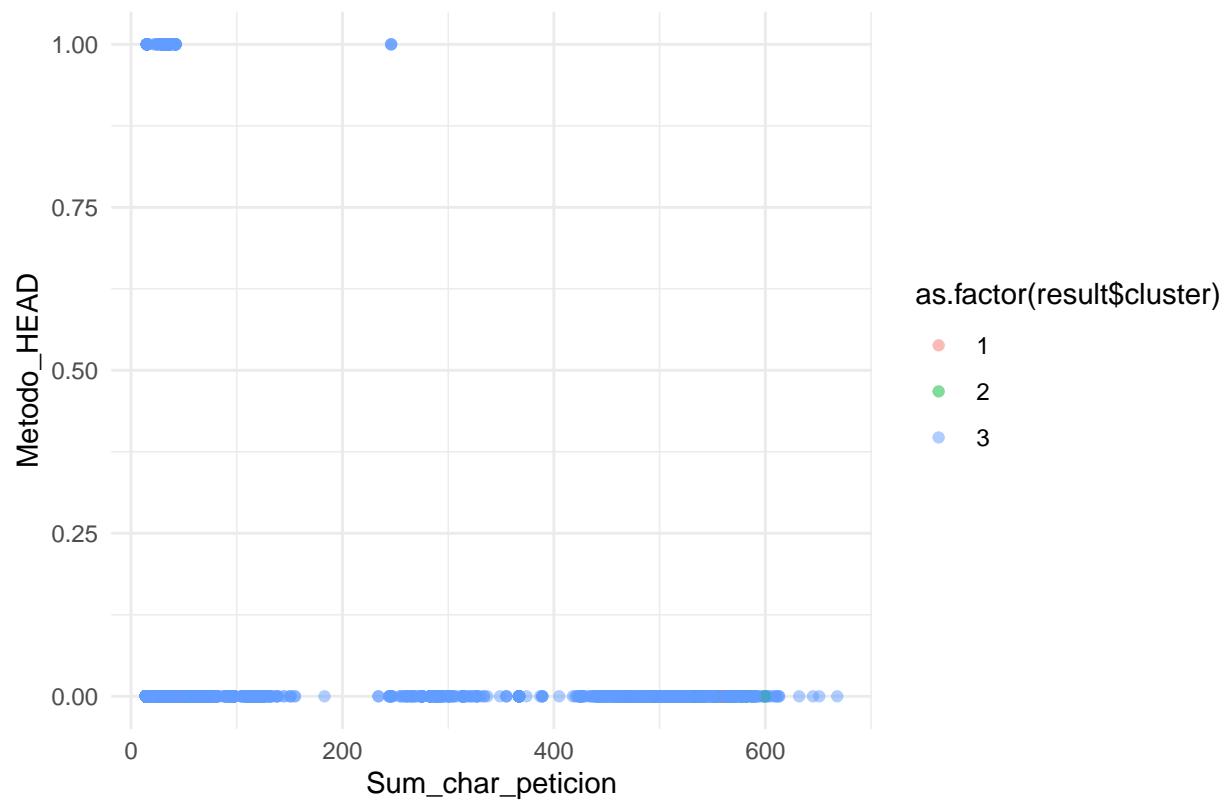




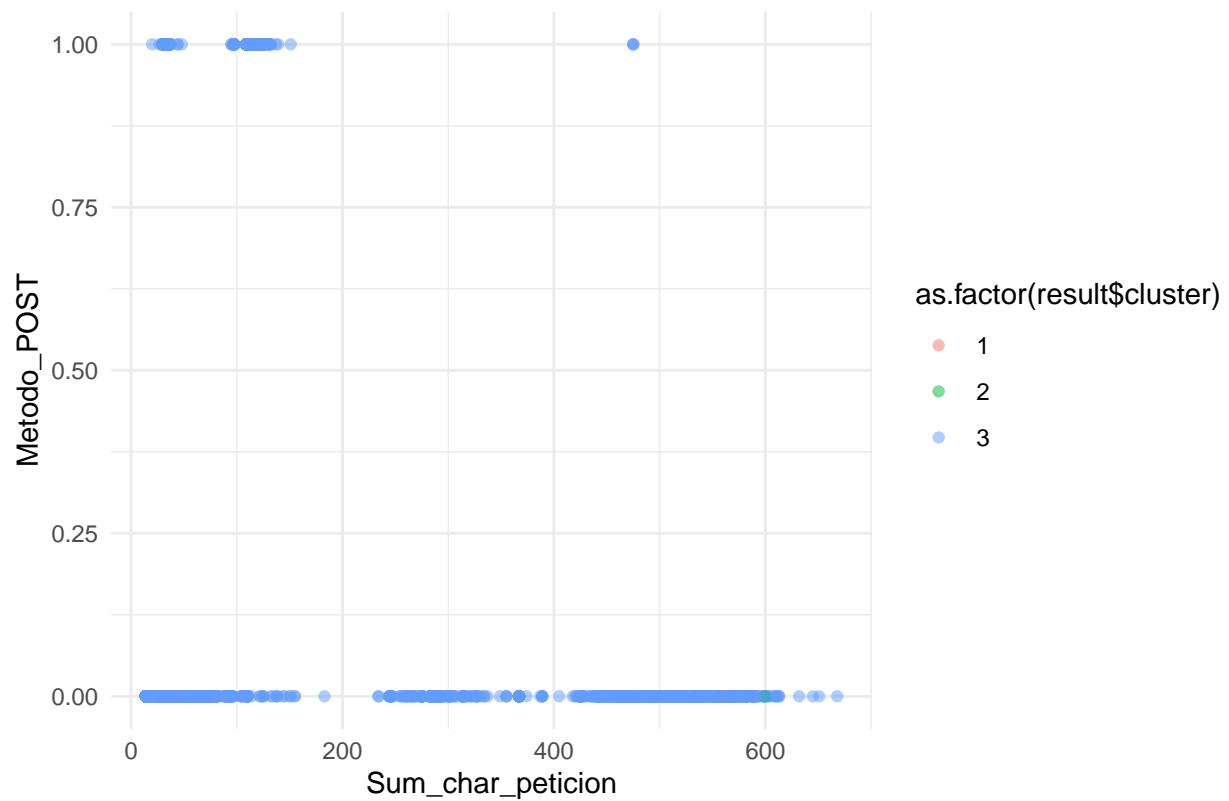
Sum_char_peticion vs Metodo_GET



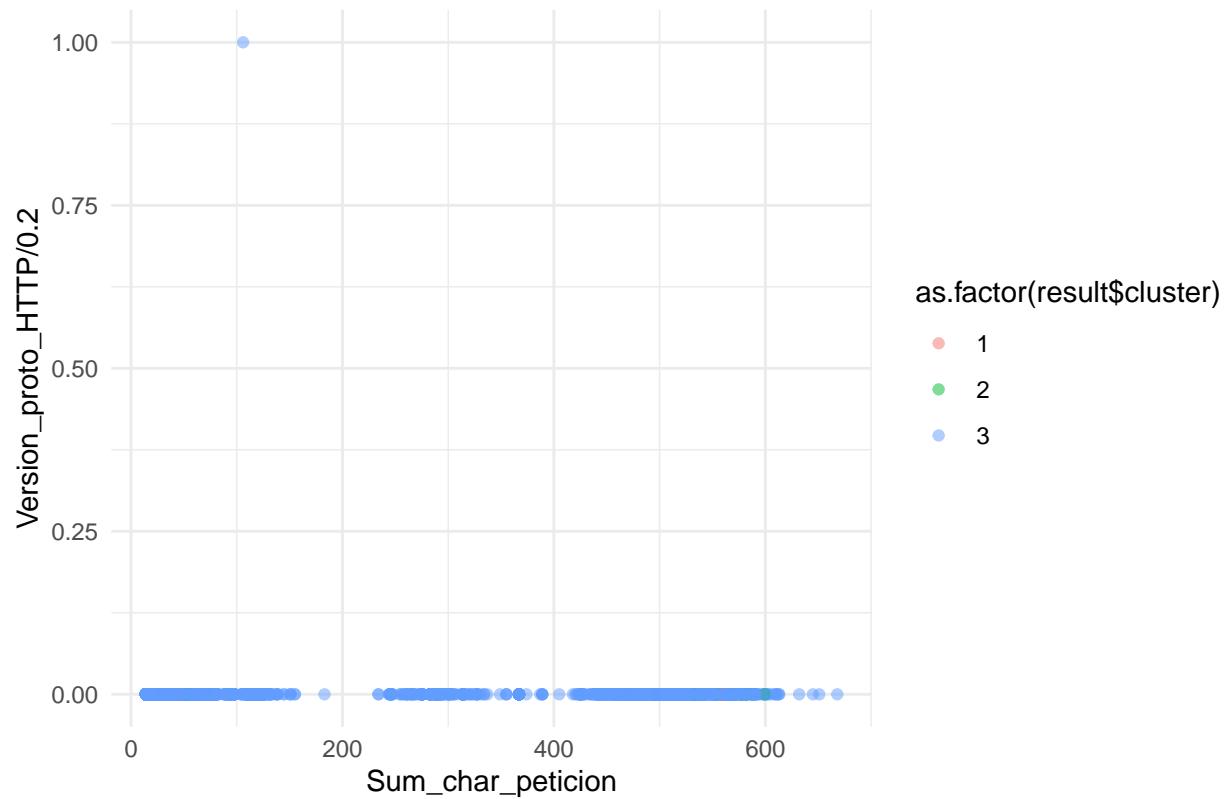
Sum_char_peticion vs Metodo_HEAD



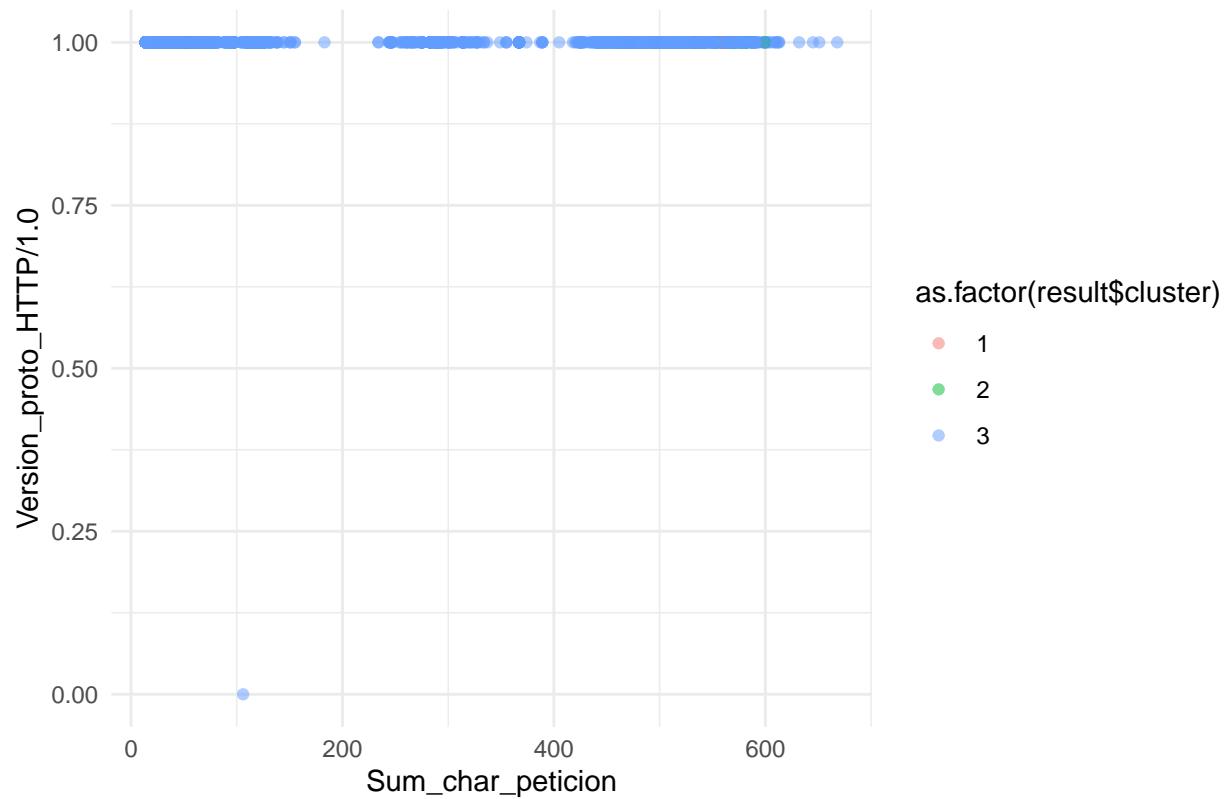
Sum_char_peticion vs Metodo_POST



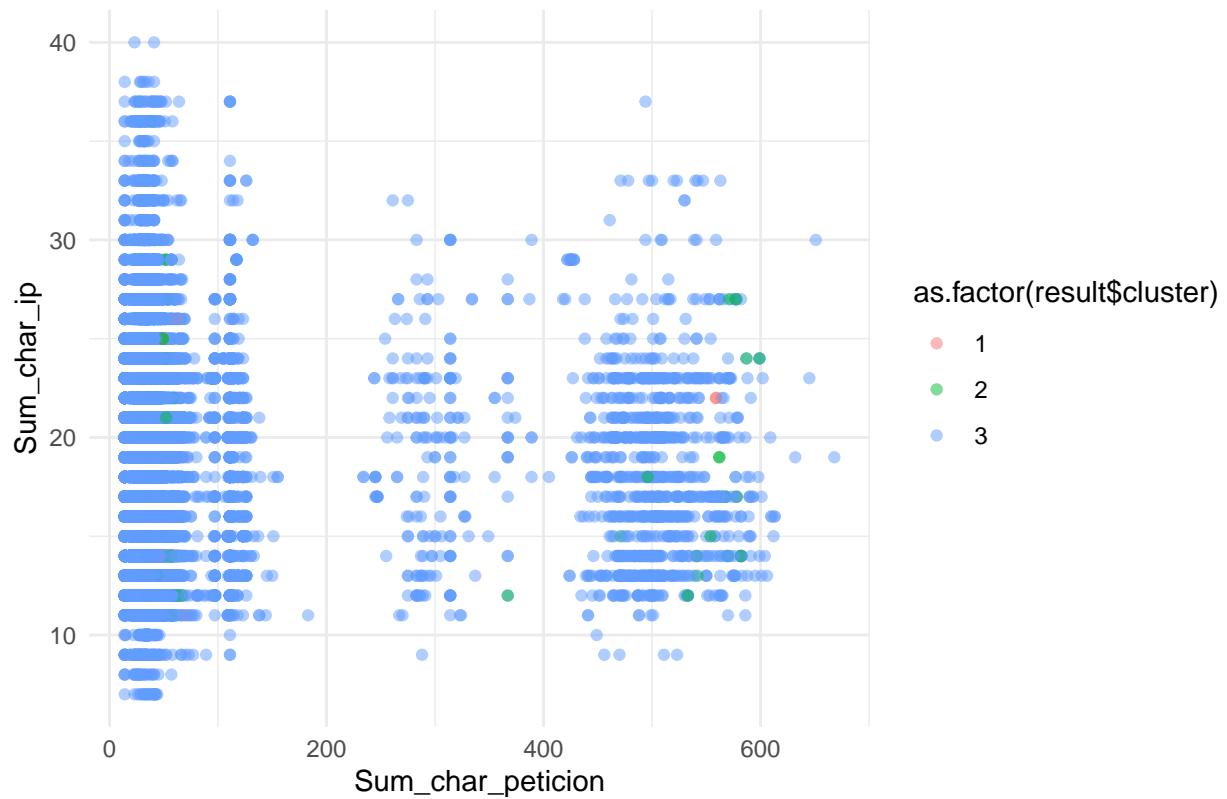
Sum_char_peticion vs Version_proto_HTTP/0.2

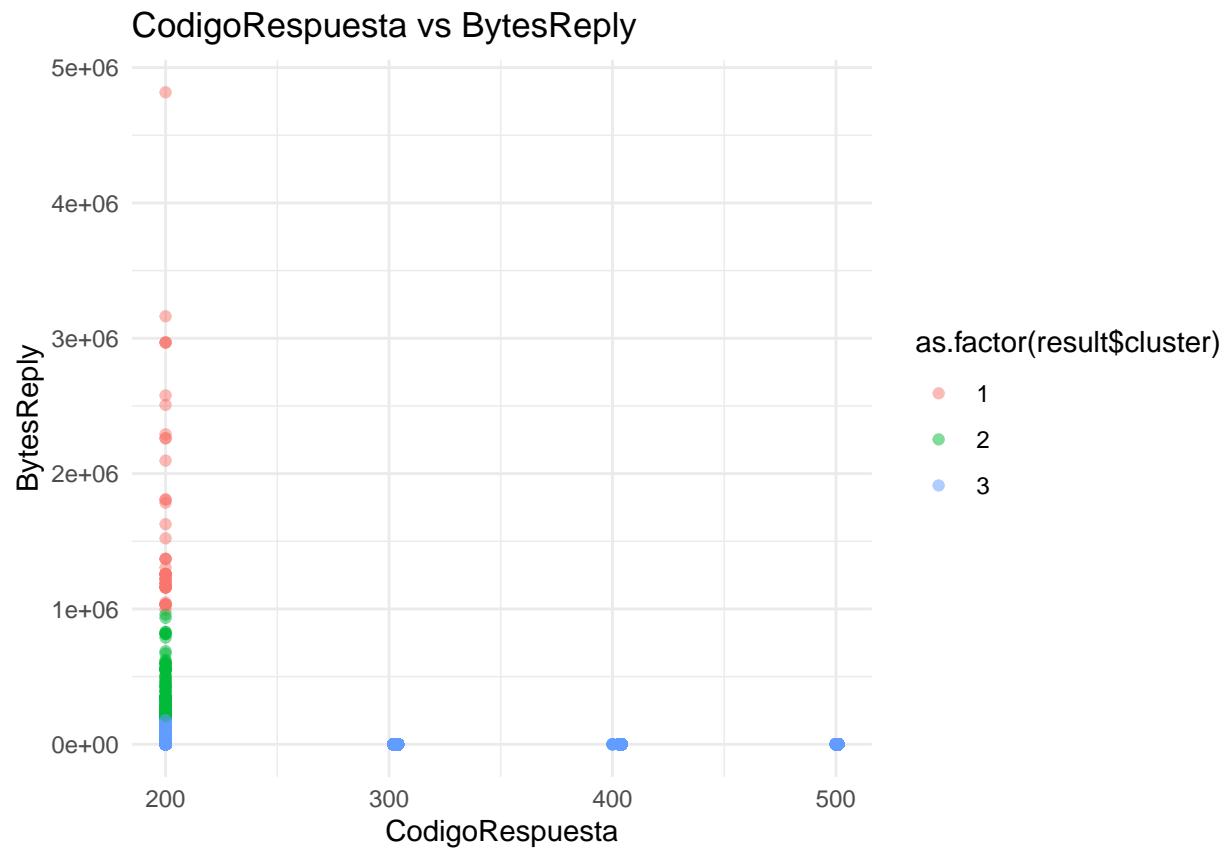


Sum_char_peticion vs Version_proto_HTTP/1.0

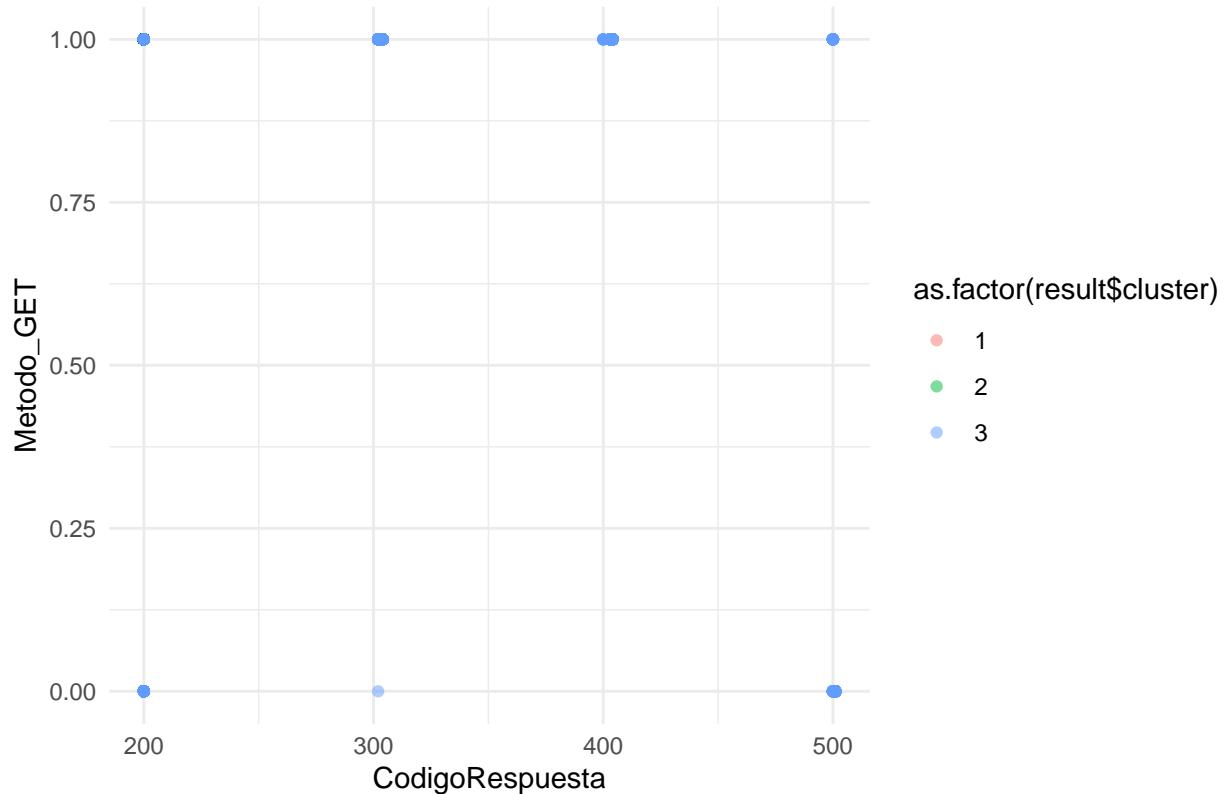


Sum_char_peticion vs Sum_char_ip

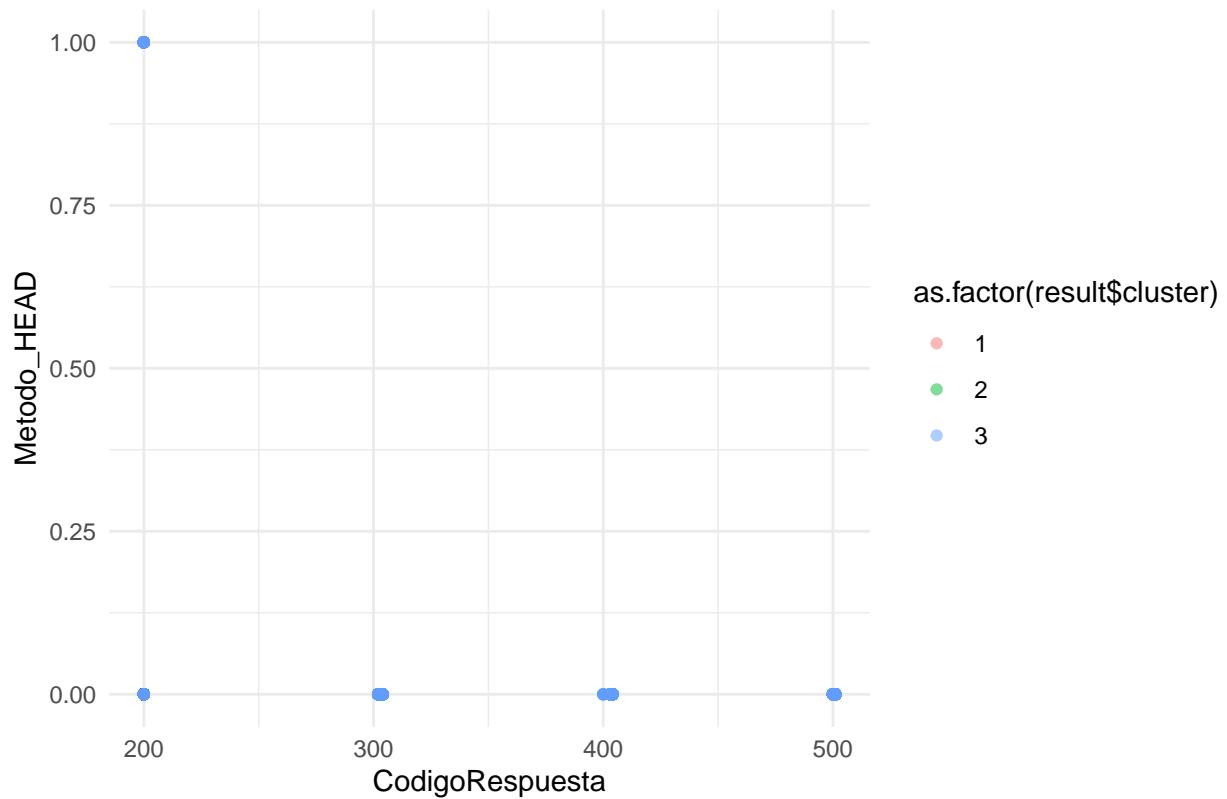




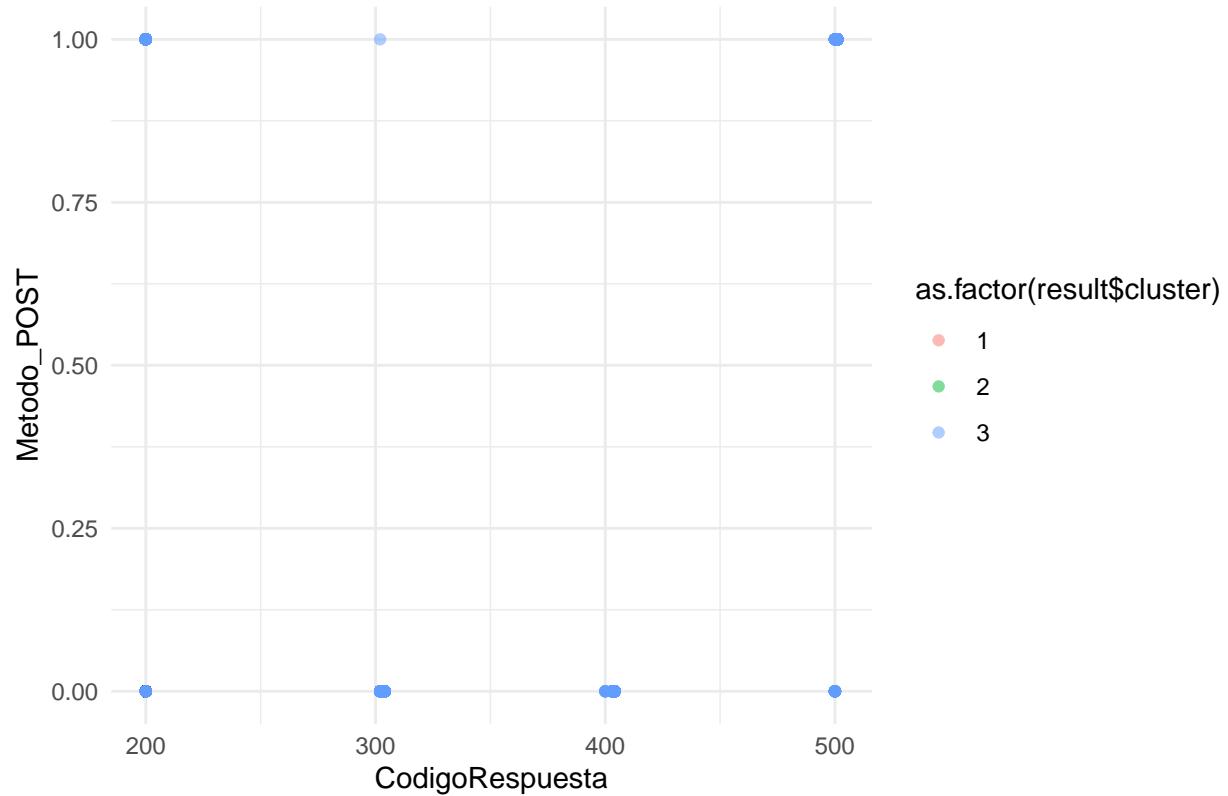
CodigoRespuesta vs Metodo_GET



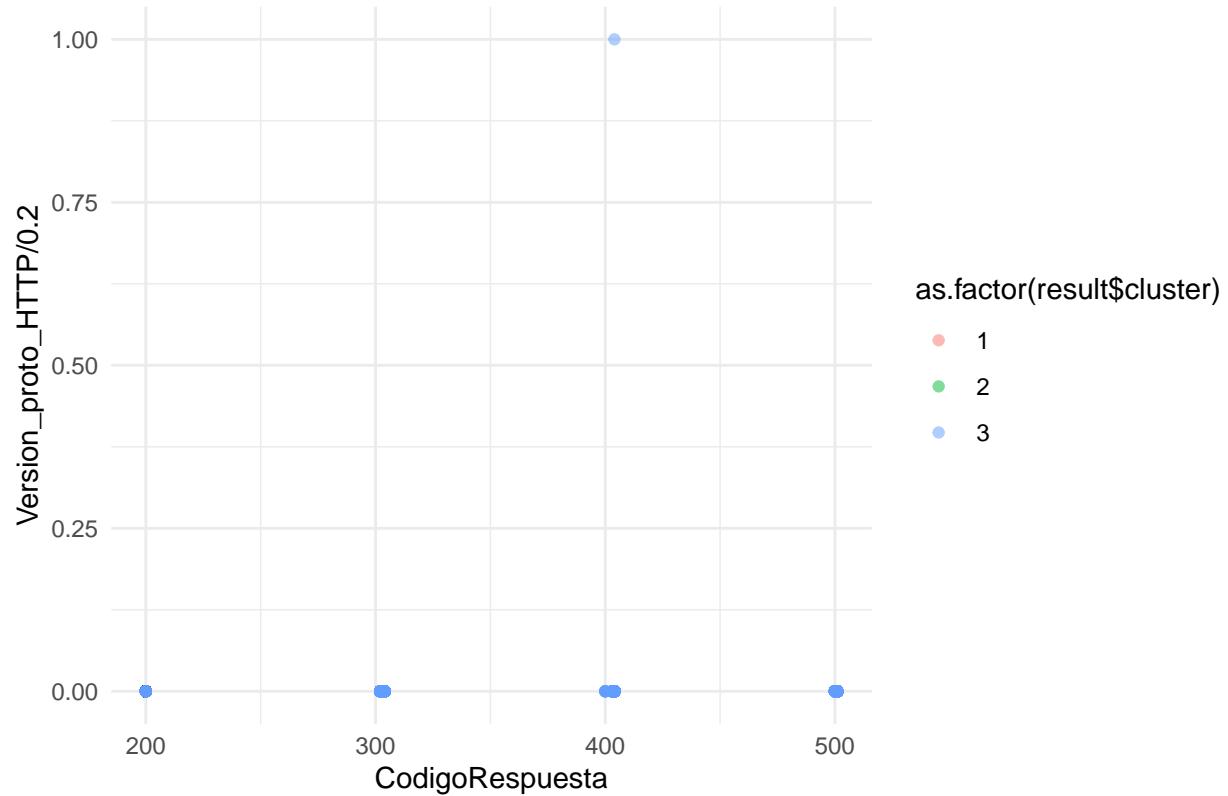
CodigoRespuesta vs Metodo_HEAD



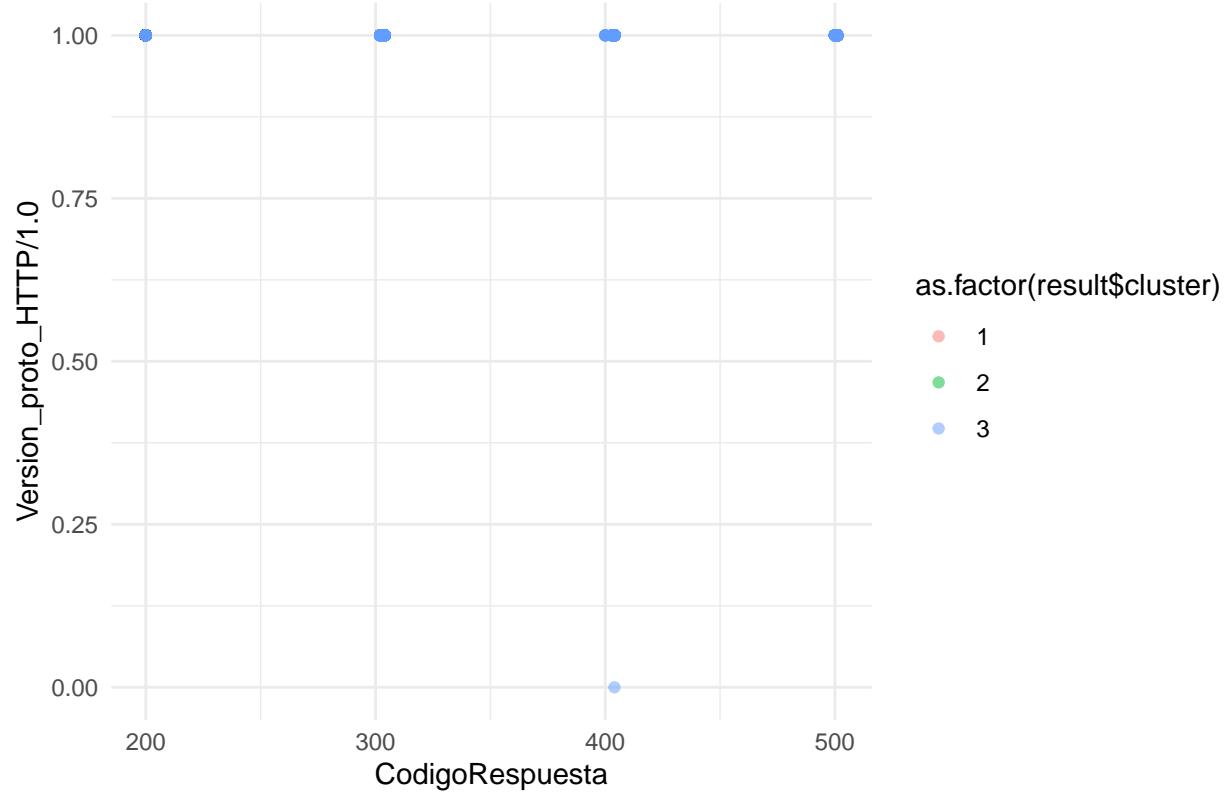
CodigoRespuesta vs Metodo_POST



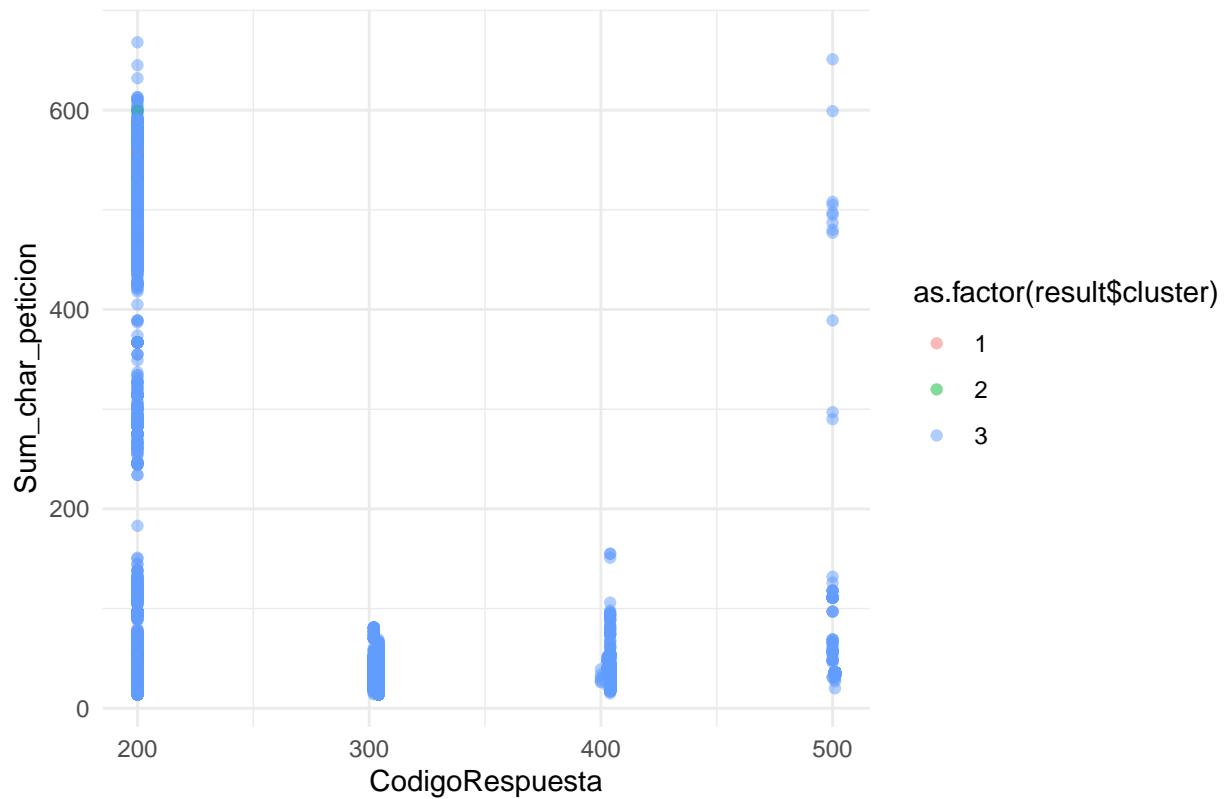
CodigoRespuesta vs Version_proto_HTTP/0.2



CodigoRespuesta vs Version_proto_HTTP/1.0



CodigoRespuesta vs Sum_char_peticion



CodigoRespuesta vs Sum_char_ip

