Caso Practico Cyclistic

Juan Ignacio Montalbán

2024-01-31

Introduccion

Debido a la gran cantidad de observaciones de cada uno de los distintos conjuntos de datos obtenidos, vamos a trabajar en R para ordenar y filtrar informacion relevante de manera comoda. Una vez hayamos conseguido nuestro objetivo, vamos a crear un *Script* con una serie de funciones para realizar las mismas acciones a cada uno de los conjuntos de datos. Despues, vamos a flitrar los conjuntos de datos y asi pasar uno a uno los conjuntos filtrados a las hojas de calculo, de esta manera aprovecharemos lalas tablas dinamicas para realizar visualizaciones de manera rapida de los datos mes a mes.

A partir de aqui pueden ocurrir dos casos: 1. Podemos encontrarnos con el caso de que la hoja de calculo siga siendo incapaz de trabajar con la cantidad de datos que tenemos, lo cual nos llevara a tener que realizar las visualizaciones tambien con R. 2. Las hojas de calculo funcionan correctamente por lo que podremos tener una vision general rapida de como se comportan nuestros datos.

Una vez realizda una vista preeliminar, podremos unificar los diferentes conjuntos de datos en un marco de datos unicos. Esto lo podemos hacer mediante `SQL` o `R`, Lo mas probable es que utilicemos R ya que no tenemos una cuenta de BigQuery que nos pemita hacer algunas acciones pero esto lo veremos mas adelante ya que existen otras plataformas para trabajar con `SQL`.

Los siguientes pasos los explicaremos una vez vayan a ocurrir pero esta introduccion esta pensada para explicar el por que vamos a trabajar con `R`en un primer momento y que pensamos ahora mismo, a medida que nos encontremos con dificultades en el camino iremos explicando que es lo que vamos a hacer para solucionarlo.

Primeros pasos

Ya hemos previsualizado los datos viendo los archivos .csv. Se tratan de archivos sumamente grande, teniendo mas de **190.000** filas los archivos mas pequeños. Por lo tanto, vamos a cargar los archivos `.CSV` aqui para trabajar con ellos.

1.Lo primero que vamos a hacer es instalar y cargar los paquetes 'Tidyverse' y 'lubridate'.

```
# install.packages('tidyverse')
# install.packages('lubridate')
# install.packages('modeest')
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     1.3.1
## 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 become error
```

```
library(lubridate)
library(modeest)
```

Hemos puesto la instalacion de paquetes como comentarios ya que ya estaban instalados en este dispositivo.

2.Cargar 1 archivo del conjunto de datos

Para cargar un archivo en formato `.CSV` vamos a utilizar el paquete `readr` que se encuentra dentro de `tidyverse`, como ya hemos llamado a `tidyverse` no sera necesario volver a hacerlo.

Dentro del paquete `readr` se encuentra la funcion `read_csv` la cual vamos a utilizar, asignando la lectura a una variable.

Los datos se encuentran en una carpeta dentro del directorio del proyecto, llamada **Conjunto_Datos**, una vez dentro existen 12 archivos diferentes, cada uno con la informacion de cada mes del año.

Para cargar el primer archivo 'Tripsdata_2023_12.csv' vamos a tener que escribir dentro de la funcion la ruta relativa para entrar dentro de la carpeta, cargamos este archivo ya que es el mas reciente de todos.

```
trips_2023_12 <- read_csv('./Conjunto_Datos/Tripdata_2023_12.csv')
```

```
## Rows: 224073 Columns: 13
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (7): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (4): start_lat, start_lng, end_lat, end_lng
## dttm (2): started_at, ended_at
##
## 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.
```

3.Comprobar que se ha cargado bien el archivo

Viendo que ya hemos cargado el archivo, vamos a comprobar que se visualiza de manera correcta y si se visualiza correctamente realizaremos el mismo paso para el resto de achivos. No lo mostraremos aqui pero para visualizarlo podemos usar la funcion `View` (importante la V en mayuscula).

Debido a la cantidad de datos que tenemos, vamos a cambiar el orden de ejecucion, vamos a realizar tanto las comprobaciones en un solo conjunto de datos y una vez lo hayamos realizado replicaremos el resto de datos y unificaremos todos los archivos para hacer comprobaciones de si existen variaciones mes a mes en el uso de Cyclistic.

Comenzamos a tratar el conjunto de datos

El siguiente paso, una vez nos hemos asegurado que todos los datos han sido comprobados, es comenzar a tratar los datos para el analisis. vamos a comprobar si podemos usar la informacion de las columnas 'start_station_name', 'start_station_id', 'end_station_name' y 'end_station_id', Ya que hemos visto algunos valores ausentes en esas columnas, vamos a comprobar estos valores para todas las columnas.

```
# Este codigo utiliza la funcion apply para poder sacar una tabla de la suma de valores faltantes por c
# X es la matriz de entrada, en este caso el dataframe completo, MARGIN puede tomar valores 1 para fila
# Fun es la funcion que le aplicaremos a cada fila, en este caso sera suma para conocer el numero de fi
# tambien podemos realizar la media para conocer el porcentaje

# Vamos a seguir usardo trips_2023_12 ya que es el mas reciente de todos y cuando acabemos de tratar lo

apply( X = is.na(trips_2023_12), MARGIN = 2, FUN = sum)
```

```
##            ride_id      rideable_type         started_at           ended_at
##                  0                  0                  0                  0
## start_station_name   start_station_id   end_station_name     end_station_id
##              35710              35710              37924              37924
##          start_lat          start_lng            end_lat            end_lng
##                  0                  0                239                239
##      member_casual
##                  0
```

```
apply( X = is.na(trips_2023_12), MARGIN = 2, FUN = mean)
```

```
##            ride_id      rideable_type         started_at           ended_at
##        0.000000000        0.000000000        0.000000000        0.000000000
## start_station_name   start_station_id   end_station_name     end_station_id
##        0.159367706        0.159367706        0.169248415        0.169248415
##          start_lat          start_lng            end_lat            end_lng
##        0.000000000        0.000000000        0.001066617        0.001066617
##      member_casual
##        0.000000000
```

Como podemos ver, el 15% aproximadamente de las observaciones tanto en los nombres de las estaciones como en los id son valores faltantesn ante esto podriamos hacer varias cosas pero ante el poco tiempo que tengo, las mas eficientes bajo mi punto de vista son 2: 1. Retiramos todas las filas que tengan valores faltantes, ya que se trata de un 15% pero siguen siendo un gran numero de observaciones con las que nos quedariamos. 2. No contamos con dichas columnas a la hora de realizar estudios sobre los datos o contamos con ellas lo minimo posible. Realmente no sabemos si en retirar esas filas nos podria provocar un sesgo al retirarlas, al no conocer si los datos faltan debido a un error y es de manera aleatoria o si se trata siempre del mismo motivo. Retirar esas columnas nos podria estar retirando observaciones valiosas lo cual nos podria provocar que lleguemos a unos resultados erroneos durante nuestro analisis.

Si tuviesemos mas tiempo, considero que lo correcto seria contactar con la empresa Cyclistic para preguntarle acerca de los datos faltantes y asi decidir que podemos hacer, si retirar todas las filas o por ejemplo generar esos datos si es que podemos mediante observacion de columnas como las coordenadas de las escaciones ya que solo existe un 0.1% de valores faltantes. En cualquier caso debido a que no tenemos suficiente tiempo como para llevar a cabo este proceso, vamos a seguir adelante teniendo los valores faltantes pero evitando utilizar en la medida de lo posible las columnas de las estaciones.

1.Inspeccionamos el conjunto

Ahora que hemos decidido como seguir, vamos a inspeccionar los valores del conjunto y decidir que datos debemos añadir (por ejemplo operando entre columnas). Ademas, Voy a comprobar los valores unicos existentes en rideable_type y member_casual ya que deberian ser solo 2 en cada columna: Electric bike o classic para la primera y member o casual en la segunda columna ya que summary no me proporcionara esa informacion y necesito saber si es correcto antes de operar.

```
### Comprobacion de informacion en las columnas
summary(trips_2023_12) # informacion de los datos en las columnas,
```

```
##    ride_id          rideable_type        started_at
##  Length:224073      Length:224073      Min.   :2023-12-01 00:00:03.00
##  Class :character   Class :character   1st Qu.:2023-12-07 16:18:35.00
##  Mode  :character   Mode  :character   Median :2023-12-13 12:05:44.00
##                                        Mean   :2023-12-14 08:30:56.74
##                                        3rd Qu.:2023-12-20 14:14:23.00
##                                        Max.   :2023-12-31 23:59:38.00
##
##     ended_at                      start_station_name start_station_id
##  Min.   :2023-12-01 00:04:12.00   Length:224073      Length:224073
##  1st Qu.:2023-12-07 16:30:49.00   Class :character   Class :character
##  Median :2023-12-13 12:16:31.00   Mode  :character   Mode  :character
##  Mean   :2023-12-14 08:44:20.97
##  3rd Qu.:2023-12-20 14:28:48.00
##  Max.   :2024-01-01 23:50:51.00
##
##  end_station_name   end_station_id       start_lat       start_lng
##  Length:224073      Length:224073      Min.   :41.65   Min.   :-87.85
##  Class :character   Class :character   1st Qu.:41.88   1st Qu.:-87.66
##  Mode  :character   Mode  :character   Median :41.90   Median :-87.64
##                                        Mean   :41.90   Mean   :-87.65
##                                        3rd Qu.:41.93   3rd Qu.:-87.63
##                                        Max.   :42.07   Max.   :-87.53
##
##     end_lat         end_lng       member_casual
##  Min.   :41.65   Min.   :-87.96   Length:224073
##  1st Qu.:41.88   1st Qu.:-87.66   Class :character
##  Median :41.90   Median :-87.64   Mode  :character
##  Mean   :41.90   Mean   :-87.65
##  3rd Qu.:41.93   3rd Qu.:-87.63
##  Max.   :42.07   Max.   :-87.53
##  NA's   :239     NA's   :239
```

```
# unique(trips_2023_12$rideable_type) # Comprobacion de los valores en la columna rideable_type y saber
# unique(trips_2023_12$member_casual) # Comprobacion de los valores en la columna member_casual y saber

### Comprobacion de los numeros y la frecuencia de los valores en la columna member_casual una vez nos
# table(trips_2023_12$member_casual) # muestra la cantidad de veces de cada una de las observaciones
prop.table(table(trips_2023_12$member_casual))# prop.table nos da la frecuencia de la proporcion de la
```

```
##
##    casual    member
## 0.2306034 0.7693966
```

```
                                                # Es decir, nos muestra la relacion de numero de veces qu
```

La informacion obtenida mediante la funcion `summary` es util no por la informacin en si, que no son realmente utiles los datos debido a la falta de columnas numericas como tal, sino que nos permiten conocer hacia donde o hacia donde no continuar trabajando en este caso concreto.

Como hemos visto que son correctos los valores de cada columna mediante `unique`, comprobamos la proporcion de los valores en la columna member_casual: Para el mes de diciembre de 2023, el 23% de los clientes era casual y casi el 77% eran miembros. deberemos comprobar si esto es asi durante todo el año o si se trata de una ocurrencia puntual. Tendremos que comprobar para cada mes esta misma informacion, por ello nos facilitara el trabajo crear las columnas mencionadas anteriormente. La columna de año la vamos a crear para poder realizar este analisis en el futuro y observar la tendencia.

Vamos a crear nuevas columnas: 'day', 'month', 'year', 'day_of_week'. Que nos den mas informacion y podamos realizar calculos con ellas en los siguientes pasos, ya que la informacion obtenida ahora es util pero insuficiente para realizar el analisis. Tambien crearemos 'trip_duration' (esta columna existia previamente en conjuntos de años anteriores y nos sera util para comprobar si existen diferencias entre los tipos de cliente en este area) para ver la si existe alguna relacion o patron entre el tipo de membresia y cuanto tiempo duran los viajes.

2.Creacion de nuevas columnas

vamos a crear nuevas columnas mediante el uso del paquete `lubridate` y mediremos el tiempo de cada viaje para poder realizar calculos con ello.

```
trips_2023_12$date <- as.Date(trips_2023_12$started_at)
trips_2023_12$day <- format(as.Date(trips_2023_12$date),'%d')
trips_2023_12$month <- format(as.Date(trips_2023_12$date),'%m')
trips_2023_12$year <- format(as.Date(trips_2023_12$date),'%Y')
trips_2023_12$day_of_week <- format(as.Date(trips_2023_12$date),'%A')
```

Una vez comprobado que se han creado correctamente las columnas de fecha, vamos a crear una columna que nos de la duracion del viaje

```
trips_2023_12$trip_duration <- difftime(trips_2023_12$ended_at,trips_2023_12$started_at)
# Esto generara horas negativas para viajes que empezaron en una fecha y terminaron en fecha diferente,
```

Los datos de la columna trip_duration estan en segundos, ahora solucionaremos los errores en el calculo de tiempo.

Observamos todas las entradas en las cuales el tiempo de trayecto es negativo, puede ocurrir que si se empieza un trayecto a ultimas horas de un dia y se finaliza el trayecto a primeras horas del ultimo dia, la duracion puede dar negativa. En nuestro caso solo existen 10 filas de datos con duraciion de trayecto negativa y en todos los casos se trata de pocos segundos y en la misma fecha, por lo que podemos eliminar esas entradas ya que parecen ser errores del sistema y no nos cambiara el resultado practicamente nada.

Para realizar la eliminacion de las filas correspondientes utilizamos la función `filter`, la cual nos permite eliminar las filas erroneas mediante una condicion. Utilizamos `filter` en lugar de `subset` debido al tamaño de nuestra muestra ya que es mas efficiente la función del paquete `dplyr`.

```
trips_2023_12_v2 <- filter(trips_2023_12, trip_duration>0)
```

Ahora vamos a convertir la columna trip_duration en numerico, para que podamos realizar calculos con ella. para ello vamos a usar la funcion `as.numeric` ya que nos permite convertir nuestra columna del tipo drifftime

a tipo numeric, si fuese de tipo `factor` seria necesario pasar primero la columna a tipo `character` antes de pasar a `numeric`. Esto es debido a que cuando un vector con numeros tiene como tipo el factor, lo que nos devuelve la funcion `as.numeric` es los valores del nivel interno del factor, es decir, el valor de prioridad equivalente al valor dentro de ese vector. Como nosotros no queremos saber en que posicion dentro de la preferencia de orden se encuentran nuestros numeros, sino que queremos el valor en si, tendremos que pasar primero la columna al tipo `charactery` despues esta columna a tipo `numeric`. todo esto no sera necesario para nosotros ya que el tipo de nuestra columna es `difftime` y realizaremos la accion descrita al principio de este parrafo.

```
trips_2023_12_v2$trip_duration <- as.numeric(trips_2023_12_v2$trip_duration)
```

Comenzamos el analisis descriptivo

Ahora vamos a comenzar a realizar el analisis descriptivo, para ello vamos a utilizar la funcion `summary` en la columna 'trip_duration' ya que esto nos realziara los calculos que queremos ver como son la media, la mediana, el valor maximo y el valor minimo. Ademas de esto, realizaremos tambien la moda mediante el paquete `modest` para conocer que tiempo de viaje es el mas comun.

```
summary(trips_2023_12_v2$trip_duration) # Usamos summary ya que realiza los calculos que queremos, adem
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     1.0   281.0   462.0   804.4   780.0 89997.0
```

```
mfv(trips_2023_12_v2$trip_duration) # la funcion mfv nos devuelve la moda de la columna
```

```
## [1] 276
```

Como vemos, los valores de la columna 'trip_duration' estan en segundos y con las funciones que acabamos de usar podemos tener una vista general de como se comportan nuestros datos por toda la columna.

```
aggregate(trips_2023_12_v2$trip_duration ~ trips_2023_12_v2$member_casual, FUN = mean)
```

```
##   trips_2023_12_v2$member_casual trips_2023_12_v2$trip_duration
## 1                         casual                      1196.4075
## 2                         member                       686.9441
```

```
aggregate(trips_2023_12_v2$trip_duration ~ trips_2023_12_v2$member_casual, FUN = median)
```

```
##   trips_2023_12_v2$member_casual trips_2023_12_v2$trip_duration
## 1                         casual                           507
## 2                         member                           450
```

```
# aggregate(trips_2023_12_v2$trip_duration ~ trips_2023_12_v2$member_casual, FUN = max)
# aggregate(trips_2023_12_v2$trip_duration ~ trips_2023_12_v2$member_casual, FUN = min)
```

No existen difernecia entre los miembros o los ciclistas casuals para los valores maximos o minimos a lo largo del mes ya que existen muchos datos, de igual forma no conocemos la informacion de si existe una diferencia de comportamiento para cada dia de la semana como si vemos a lo largo del mes. A lo largo del mes comprobamos que existe una diferencia ya que los clientes casuales tienen de media casi el doble de duracion en sus viajes que los miembros.

```r
trips_2023_12_v2$day_of_week <- ordered(trips_2023_12_v2$day_of_week, levels = c("lunes","martes","miér
aggregate(trips_2023_12_v2$trip_duration ~ trips_2023_12_v2$member_casual + trips_2023_12_v2$day_of_weel
```

```
##    trips_2023_12_v2$member_casual trips_2023_12_v2$day_of_week
## 1                          casual                       lunes
## 2                          member                       lunes
## 3                          casual                      martes
## 4                          member                      martes
## 5                          casual                   miércoles
## 6                          member                   miércoles
## 7                          casual                      jueves
## 8                          member                      jueves
## 9                          casual                     viernes
## 10                         member                     viernes
## 11                         casual                      sábado
## 12                         member                      sábado
## 13                         casual                     domingo
## 14                         member                     domingo
##    trips_2023_12_v2$trip_duration
## 1                       1133.4539
## 2                        691.8619
## 3                        987.0701
## 4                        643.0881
## 5                       1067.9067
## 6                        652.0568
## 7                       1016.3864
## 8                        693.4585
## 9                       1068.3369
## 10                       690.5248
## 11                      1341.0736
## 12                       705.7825
## 13                      1667.1422
## 14                       750.1778
```

```r
# aggregate(trips_2023_12_v2$trip_duration ~ trips_2023_12_v2$member_casual + trips_2023_12_v2$day_of_w
# ordenamos la columna de dias de la semana de lunes a domingos, y con la funcion `agreggate` realizamo
# columna despues del signo ~ ,
```

Como podemos observar los clientes casuales realizan viajes con mayor duracion (casi el doble) que los miembros. siendo los dias de mas duracion sabado y domingo y siendo el dia de menos duracion los martes.

```r
trips_2023_12_v2 %>%
  mutate(weekday = wday(started_at, label = TRUE, abbr = FALSE, week_start=1)) %>%    # crea una columna
  group_by(member_casual, weekday) %>%                     # agrupamos por member_casual y weekday
  summarise(number_of_rides = n()                                    # calcula el numero de viajes
  ,average_duration = mean(trip_duration)) %>%               # calcula la media de duracion de viajes
  arrange(member_casual, weekday)                            # ordena
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 14 x 4
```

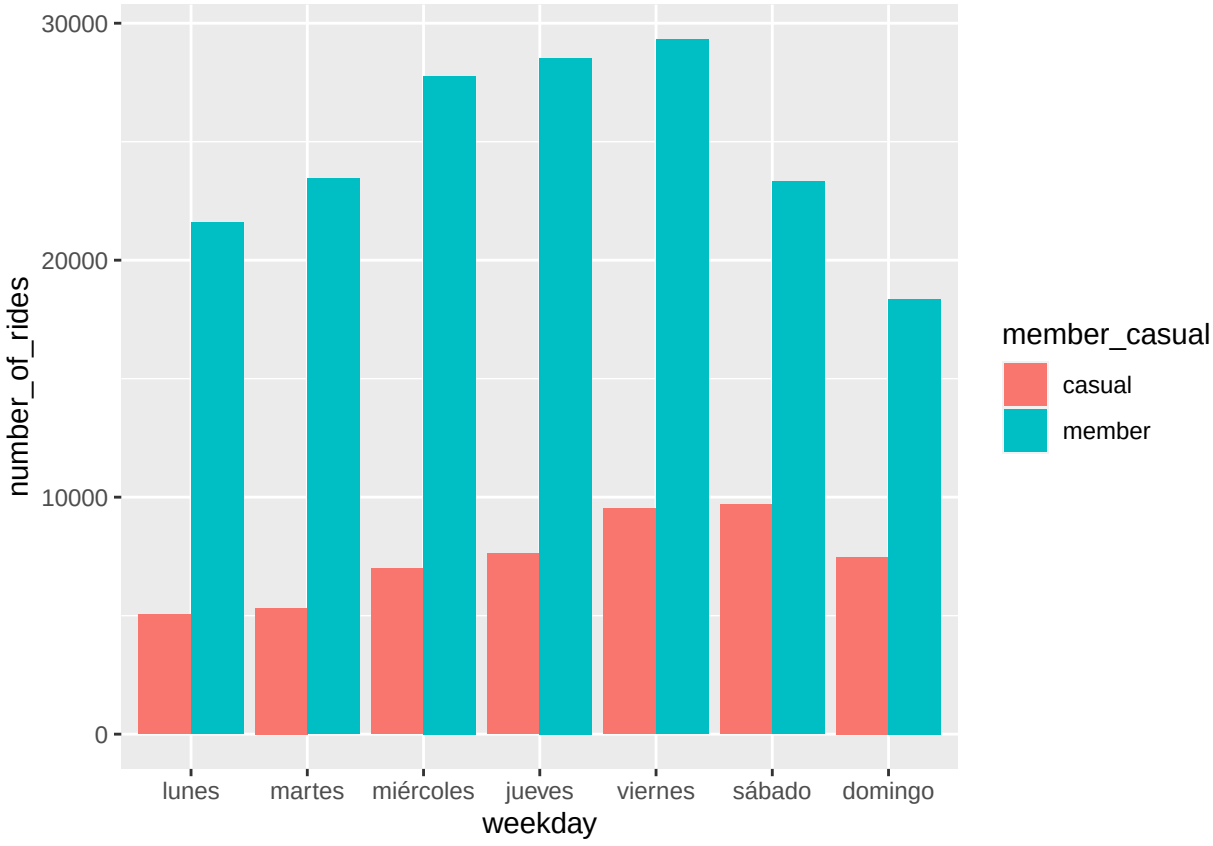
```
## # Groups:   member_casual [2]
##    member_casual weekday   number_of_rides average_duration
##    <chr>         <ord>               <int>            <dbl>
##  1 casual        lunes                5045            1133.
##  2 casual        martes               5319             987.
##  3 casual        miércoles            6979            1068.
##  4 casual        jueves               7637            1016.
##  5 casual        viernes              9511            1068.
##  6 casual        sábado               9698            1341.
##  7 casual        domingo              7473            1667.
##  8 member        lunes               21593             692.
##  9 member        martes              23454             643.
## 10 member        miércoles           27774             652.
## 11 member        jueves              28534             693.
## 12 member        viernes             29319             691.
## 13 member        sábado              23318             706.
## 14 member        domingo             18364             750.
```

Creacion de graficos

Aqui vemos la diferencia entre los clientes, aunque se puede apreciar cierta diferencia entre los tipos, vamos a utilizar un grafico para poder visualizar con mayor claridad las diferencias, especialmente segun varian los dias de la semana.

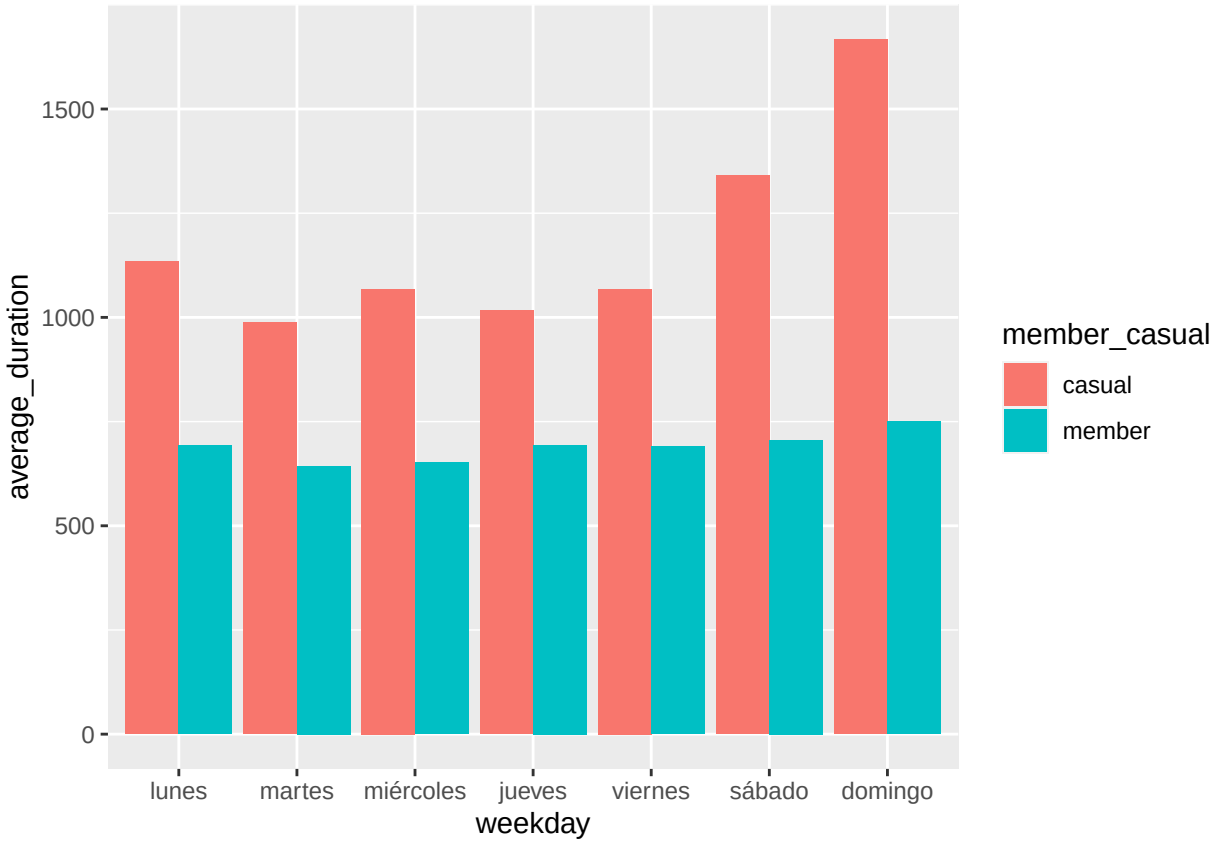
```
trips_2023_12_v2 %>%
  mutate(weekday = wday(started_at, label = TRUE, abbr = FALSE, week_start=1)) %>%    # crea una columna
  group_by(member_casual, weekday) %>%                     # agrupamos por member_casual y weekday
  summarise(number_of_rides = n()                                    # calcula el numero de viajes
  ,average_duration = mean(trip_duration)) %>%                # calcula la media de duracion de viajes
  arrange(member_casual, weekday) %>%                        # ordena
  ggplot(aes(x = weekday, y = number_of_rides, fill = member_casual)) + # usa number_of_rides para el e
  geom_col(position = "dodge")
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```




```
trips_2023_12_v2 %>%
  mutate(weekday = wday(started_at, label = TRUE, abbr = FALSE, week_start=1)) %>%
  group_by(member_casual, weekday) %>%
  summarise(number_of_rides = n()
  ,average_duration = mean(trip_duration)) %>%
  arrange(member_casual, weekday)  %>%
  ggplot(aes(x = weekday, y = average_duration, fill = member_casual)) + # average_duration para el eje
  geom_col(position = "dodge")
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```



Visualizando los datos comprobamos que a pesar de que el tiempo empleado por los clientes indica que los casuales utilizan por mas tiempo las bicicletas, vemos que los miembros utilizan mas veces el servicio a lo largo del mes.

Crear funciones

Ya que hemos podido analizar el conjunto de datos de diciembre, vamos a crear unas funciones que nos traten los diferentes conjuntos para el resto de meses del año. Estas funciones las creamos para no tener que realizar cada paso seguido anteriormente una y otra vez.

```
#### AÑADE LAS COLUMNAS DE FECHA Y COLUMNA DE DURACION DEL VIAJE PARA HACER CALCULOS
add_date_columns <- function(data, datetime_column_start, datetime_column_end) {
  data %>%
    mutate(date = as.Date(data[[datetime_column_start]]), # Crea una columna date que no tiene la hora
           day = day(as.Date(data[[datetime_column_start]])), # Crea una columna tomando el dia de la c
           month = month(as.Date(data[[datetime_column_start]])), # Crea una columna tomando el mes de
           year = year(as.Date(data[[datetime_column_start]])), # Crea una columna tomando el año de la
           day_of_week = wday(as.Date(data[[datetime_column_start]]), label = TRUE, abbr = FALSE), # Cr
                              # semana, el arg abbr=FALSE es para que muestre el nombre completo
           trip_duration = difftime(data[[datetime_column_end]], data[[datetime_column_start]])) # Crea
           # de trayecto en segundos

}

#### ELIMINA LAS FILAS QUE TIENEN VALORES NEGATIVOS EN UNA COLUMNA DADA (trip_duration)
```

```r
remove_negatives <- function(data, filter_column) {
  result <- data %>%
    filter(filter_column > 0) %>% # Filtra las filas mediante valores en la columna dada, queremos usar
    # la columna trip_duration

  # Devuelve el dataframe ya modificado
  return(result)

}

#### CAMBIA EL TYPE DE LA COLUMNA DADA
change_column_type <- function(data, column_name, type = "character") {
  # Revisa que la columna existe en el dataframe
  if (!(column_name %in% names(data))) {
    stop("Columna no encontrada en el dataframe.")
  }

  # Convierte la columna especificada en el tipo elegido
  switch(type,
         "logical" = data[[column_name]] <- as.logical(data[[column_name]]), # cambia la columna a tipo
         "character" = data[[column_name]] <- as.character(data[[column_name]]), #cambia la columna a t
         "numeric" = data[[column_name]] <- as.numeric(data[[column_name]]), # cambia la columna a tipo
         stop("Error en type. Elige entre 'logical', 'character', o 'numeric'.") # Si introducimos un n
  )

  # Devuelve el dataframe ya modificado
  return(data)
}

#### FUNCION QUE UNIFICA LAS TRES YA ESCRITAS PARA SOLO LLAMAR A UNA FUNCION
funcion_unif <- function(data, datetime_column_start, datetime_column_end, filter_column, # Funcion que
                         column_to_change = NULL, new_column_type = "character") {       # column_to_c
  # Primer proceso: usa la primera funcion para crear las columnas
  data_processed <- add_date_columns(data, datetime_column_start, datetime_column_end)

  # Segundo proceso: usa la segunda funcion para filtrar
  data_processed <- remove_negatives(data_processed, filter_column)

  # Tercer proceso: if y else para saber si usar una funcion u otra
  if (!is.null(column_to_change)) {
    data_processed <- change_column_type(data_processed, column_to_change, type = new_column_type) # si
  } else {
    data_processed <- change_column_type(data_processed, filter_column, type = new_column_type) # si la

  }

  # Devuelve el dataframe modificado
  return(data_processed)
}

# Ejemplo de uso
# funcion_unif(trips_2023_01, "started_at", "ended_at", "trip_duration", new_column_type = "numeric")
```

Guardar todos los conjuntos de datos

Todo parece correcto, por lo que vamos a guardar los conjuntos de datos en variables para poder trabajar con ellos.

```
# trips_2023_02 <- read_csv('./Conjunto_Datos/Tripdata_2023_02.csv')
# trips_2023_03 <- read_csv('./Conjunto_Datos/Tripdata_2023_03.csv')
# trips_2023_04 <- read_csv('./Conjunto_Datos/Tripdata_2023_04.csv')
# trips_2023_05 <- read_csv('./Conjunto_Datos/Tripdata_2023_05.csv')
# trips_2023_06 <- read_csv('./Conjunto_Datos/Tripdata_2023_06.csv')
# trips_2023_07 <- read_csv('./Conjunto_Datos/Tripdata_2023_07.csv')
# trips_2023_08 <- read_csv('./Conjunto_Datos/Tripdata_2023_08.csv')
# trips_2023_09 <- read_csv('./Conjunto_Datos/Tripdata_2023_09.csv')
# trips_2023_10 <- read_csv('./Conjunto_Datos/Tripdata_2023_10.csv')
# trips_2023_11 <- read_csv('./Conjunto_Datos/Tripdata_2023_11.csv')
# trips_2023_01 <- read_csv('./Conjunto_Datos/Tripdata_2023_01.csv')
# trips_2023_12 <- read_csv('./Conjunto_Datos/Tripdata_2023_12.csv')
```

Este paso sirve para realizar comprobaciones previas entre los conjuntos de datos para poder unificarlos en uno solo, como y

---

1.Comprobar los nombres de las columnas

Ahora comprobaremos que los nombres de las columnas coinciden. Lo Primero comprobaremos el conjunto mas reciente, ya que si se ha realizado algun cambio a la hora de nombrar las columnas, todos los conjuntos que salgan nuevos tendran esos nombres y esa disposicion. Vamos a mirar primero el conjunto para diciembre de 2023.

```
#colnames(trips_2023_12) == colnames(trips_2023_11)
```

Tras comprobar mediante booleanos que todas las columnas en los doce archivos coinciden (su nombre, no sus valores), vamos a comprobar que las columnas de los archivos usan los mismos tipos de datos con la funcion `str`. Ej: `str(trips_2023_12)`.

---

ejecutamos los archivos y realizamos la ejecucion de las funciones que creamos previamente

---

2.Unificamos los conjuntos en un solo marco de datos

tratamos todos los conjuntos

```
#trips_2023_01_v2 <- funcion_unif(trips_2023_01, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_02_v2 <- funcion_unif(trips_2023_02, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_03_v2 <- funcion_unif(trips_2023_03, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_04_v2 <- funcion_unif(trips_2023_04, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_05_v2 <- funcion_unif(trips_2023_05, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_06_v2 <- funcion_unif(trips_2023_06, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_07_v2 <- funcion_unif(trips_2023_07, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_08_v2 <- funcion_unif(trips_2023_08, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_09_v2 <- funcion_unif(trips_2023_09, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_10_v2 <- funcion_unif(trips_2023_10, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_11_v2 <- funcion_unif(trips_2023_11, "started_at", "ended_at", "trip_duration", new_column_
#trips_2023_12_v2 <- funcion_unif(trips_2023_12, "started_at", "ended_at", "trip_duration", new_column_
```

Ahora usaremos `bind_rows` para unificar un df que tiene las mismas columnas, podriamos usar la funcion `rbind` pero ya que es menos eficiente y tenemos una cantidad de entradas considerables, vamos a utilizar bind_rows.

```
#all_trips <- bind_rows(trips_2023_12_v2, trips_2023_11_v2, trips_2023_10_v2, trips_2023_09_v2, trips_2
#                        trips_2023_06_v2, trips_2023_05_v2, trips_2023_04_v2, trips_2023_03_v2, trips_2
```

3.Guardar el nuevo archivo

Antes de comenzar a trabajar con nuestros datos, es buena idea guardar un archivo en formato `.CSV` ya que eso nos podria ayudar en el caso de que necesitemos los datos originales y no queramos tener que realizar este proceso otra vez.

```
# write_csv(all_trips,'./Conjunto_Datos/all_trips_2023.csv')
```

vamos a limpiar la memoria de nuestro entorno y volver a llamar al conjunto de datos para aliviar al equipo.

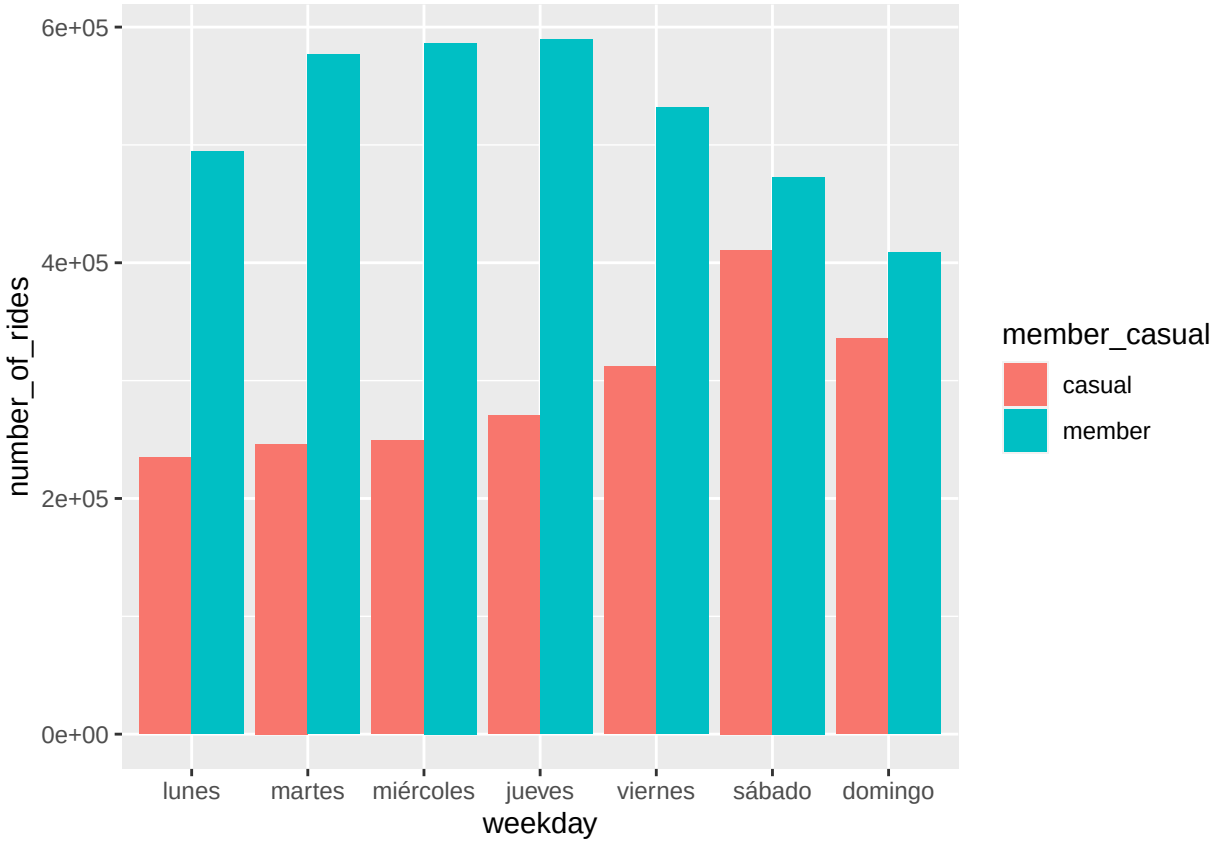
```
all_trips <- read_csv('./Conjunto_Datos/all_trips_2023.csv')
```

```
## Rows: 5719877 Columns: 19
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (8): ride_id, rideable_type, start_station_name, start_station_id, end_...
## dbl  (8): start_lat, start_lng, end_lat, end_lng, day, month, year, trip_dur...
## dttm (2): started_at, ended_at
## date (1): date
##
## 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.
```

Creamos los graficos anuales

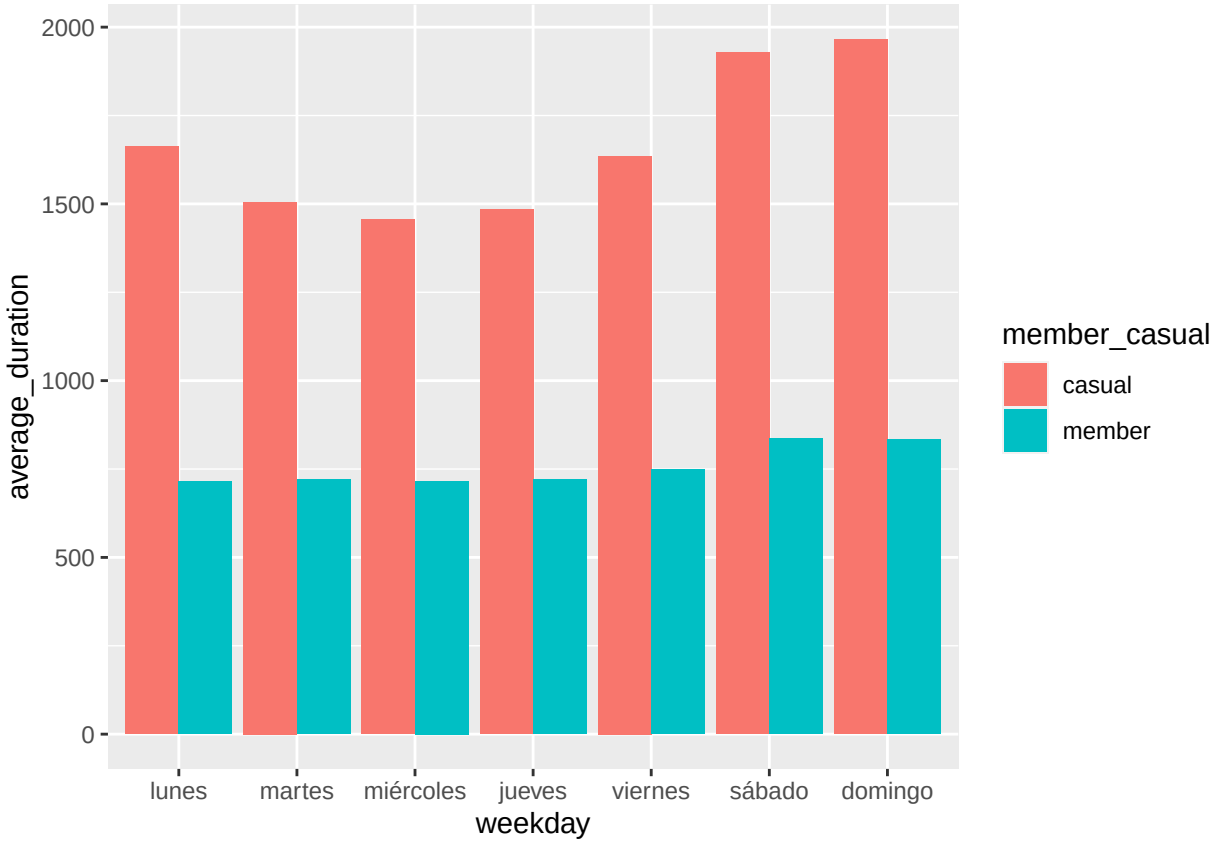
```
all_trips %>%
  mutate(weekday = wday(started_at, label = TRUE, abbr = FALSE, week_start=1)) %>%    # crea una columna
  group_by(member_casual, weekday) %>%                     # agrupamos por member_casual y weekday
  summarise(number_of_rides = n()                                    # calcula el numero de viajes
  ,average_duration = mean(trip_duration)) %>%                # calcula la media de duracion de viajes
  arrange(member_casual, weekday) %>%                        # ordena
  ggplot(aes(x = weekday, y = number_of_rides, fill = member_casual)) + # usa number_of_rides para el e
  geom_col(position = "dodge")
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```



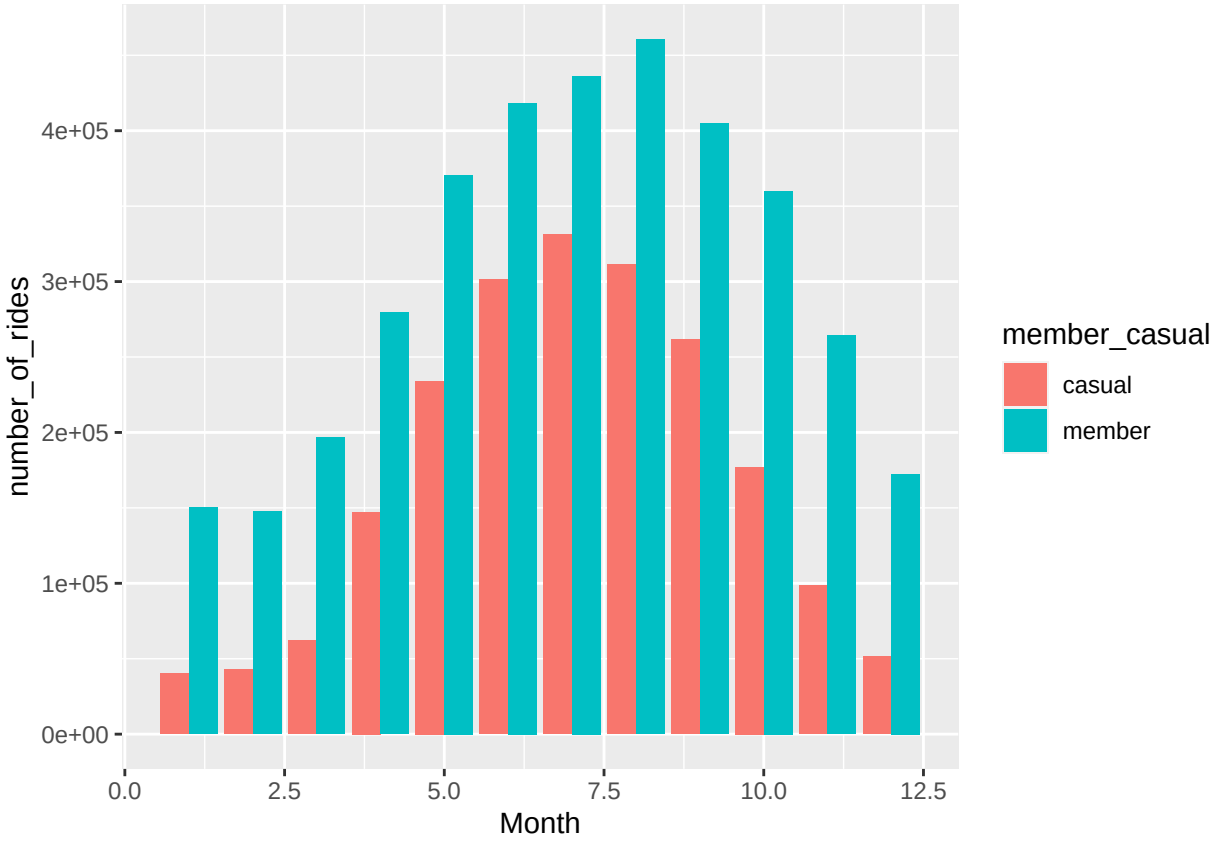

```
all_trips %>%
  mutate(weekday = wday(started_at, label = TRUE, abbr = FALSE, week_start=1)) %>%
  group_by(member_casual, weekday) %>%
  summarise(number_of_rides = n()
  ,average_duration = mean(trip_duration)) %>%
  arrange(member_casual, weekday)  %>%
  ggplot(aes(x = weekday, y = average_duration, fill = member_casual)) + # average_duration para el eje
  geom_col(position = "dodge")
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```



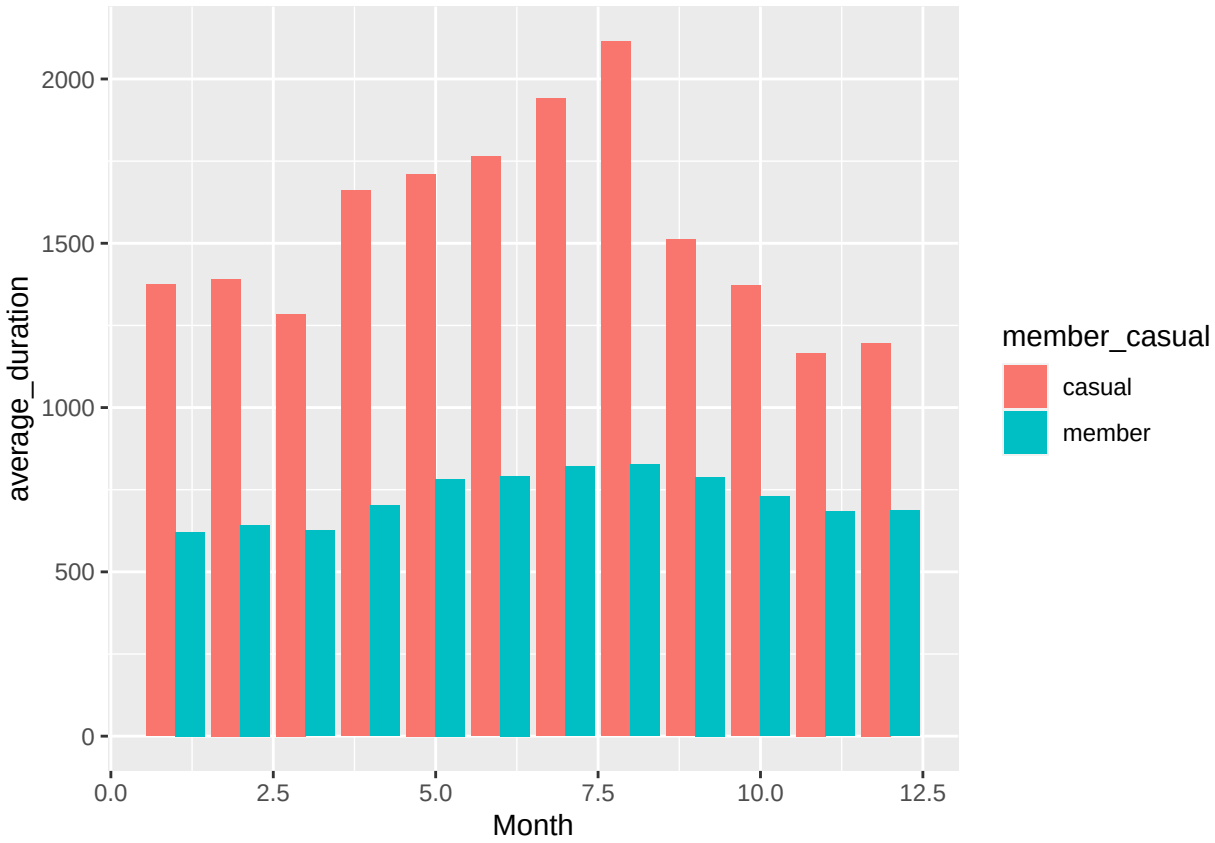

```
all_trips %>%
  mutate(Month = month(started_at)) %>%    # crea una columna usando wday()
  group_by(member_casual, Month) %>%                    # agrupamos por member_casual y weekday
  summarise(number_of_rides = n()                                  # calcula el numero de viajes
  ,average_duration = mean(trip_duration)) %>%              # calcula la media de duracion de viajes
  arrange(member_casual, Month) %>%                       # ordena
  ggplot(aes(x = Month, y = number_of_rides, fill = member_casual)) + # usa number_of_rides para el eje
  geom_col(position = "dodge")
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```




```
all_trips %>%
  mutate(Month = month(started_at)) %>%
  group_by(member_casual, Month) %>%
  summarise(number_of_rides = n()
  ,average_duration = mean(trip_duration)) %>%
  arrange(member_casual, Month)  %>%
  ggplot(aes(x = Month, y = average_duration, fill = member_casual)) + # average_duration para el eje Y
  geom_col(position = "dodge")
```

```
## `summarise()` has grouped output by 'member_casual'. You can override using the
## `.groups` argument.
```


average_duration
2000
1500
1000
500
0
0.0
2.5
5.0
7.5
10.0
12.5
Month
member_casual
casual
member