Chapter 2

MANIPULACIÓN Y VISUALIZACIÓN DE DATOS FINANCIEROS EN R

El poder importar datos en R es una tarea fundamental dentro del análisis de datos y la programación estadística. Permite cargar conjuntos de datos almacenados en diferentes formatos (como archivos CSV, Excel, bases de datos, entre otros) dentro del entorno de lenguaje R para su posterior manipulación, exploración y análisis.

Importar datos en R te permite acceder y manipular los datos de manera eficiente. Puedes realizar operaciones como filtrar, ordenar, agregar, transformar y resumir los datos según tus necesidades.

R es ampliamente utilizado para el análisis estadístico y el modelado de datos. Al importar datos en R, puedes aplicar técnicas estadísticas, realizar pruebas de hipótesis, ajustar modelos, realizar regresiones y realizar análisis exploratorio de datos.

R cuenta con una amplia variedad de paquetes y funciones para la visualización de datos. Importar datos en R te permite crear gráficos, diagramas y visualizaciones interactivas para explorar y comunicar tus resultados.

Puedes importar datos de múltiples fuentes y combinarlos para obtener un panorama completo de tus datos. R te permite fusionar, unir y combinar conjuntos de datos para realizar análisis más complejos y obtener conclusiones más sólidas.

Importar datos en R como parte de tu flujo de trabajo te ayuda a mantener la reproducibilidad. Puedes escribir scripts o documentos de R (como `R Markdown`) que contengan el código para importar los datos, lo que facilita la reproducción de tus análisis en el futuro.

2.1 IMPORTACION DE DATOS FINANCIEROS EN R

Dentro del entorno de desarrollo y trabajo de programación en R, se pueden importar datos financieros de diversas fuentes utilizando diferentes paquetes y métodos.

1. El paquete `quantmod` es ampliamente utilizado para importar datos financieros de diversas fuentes, como Yahoo Finance, Google Finance, y muchos más.

2. El paquete `tidyquant` proporciona una interfaz más fácil de usar para importar datos financieros y realizar análisis cuantitativo. puede importarse datos de precio de las acciones, índices bursátiles, tipos de interés, entre otros.

3. Si se tienen datos financieros dentro de un archivo CVS o en otro formato de archivo tabular, se puede usar la función `read.csv()` o `read.table()` en combinación con el paquete `quantmod` para importar los datos. Debe uno de estar seuro de que los datos estén en un formato adecuado (por ejemplo, conc olumnas para la fecha y los precios) y, luego, utilizar la función correspondiente para importar datos.

2.1.1 LA LIBRERÍA `quantmod`

La librería `quantmod` es una biblioteca dentro de R especializada en el análisis y modelación de datos financieros. Proporciona herramientas y funciones para la descarga de datos de mercado, cálculo de indicadores técnicos, el análisis de series de tiempo financieras, la construcción y evaluación de estrategias de inversión, entre otros.

La librería `quantmod` es un paquete de R creado por Jeffrey A. Ryan, que proporciona una amplia gama de funciones para el análisis cuantitativo de datos financieros. Está diseñado para trabajar conseries de tiempo financieras y se utiliza comúnmente en la industria financiera, la investigación académica y el trading algorítmico.

La librería `quantmod` sirve para una variedad de tareas relacionadas con el análisis de datos financieros. Algunos comunes de `quantmod` incluyen:

- Descarga de datos históricos de precios de acciones, índices y otros instrumentos financieros desde fuentes de línea.

- Cálculo de indicadores técnicos populares, como medias móviles, bandas Bollinger, MACD, RSI, entre otros.

- Análisis de series de tiempo financieras, incluyendo descomposición estacional, análisis de autocorrelación, pruebas de estacionariedad, etc.

- Construcción y evaluación de estrategias de inversión, incluyendo backtesting y simulación de cartera.

- Visualización de datos financieros en forma de diagramas y trazos de indicadores técnicos.

El desarrollo de la librería `quantmod` se basó en otras bibliotecas y paquetes de R, como la librería `xts` (para manejo de series de tiempo) y `TTR` (para el cálculo de indicadores técnicos). La combinación de estas herramientas y las funcionalidades adicionales agregadas por Ryan resultaron en la librería `quantmod` que se conoce hoy en día.

`quantmod` se ha convertido en una herramienta popular y ampliamente utilizada en el análisis cuantitativo de datos financieros en R debido a su amplia gama de funcionalidades y su facilidad de uso[1].

[1] Hay que recordar que para utilizar la librería `quantmod` en R, se debe instalar previamete utilizando la función `install.packages("quantmod")` y luego cargarla en el entorno de trabajo con la función `library(quantmod)`.

2.1.2 LA LIBRERÍA tidyquant

La librería **tidyquant** fue creada por Matt Dancho y Davis Vaughan. **tidyquant** es una herramienta popular en el análisis financiero en R que combina los paquetes **quantmod**, **TTR**, **xts**, **zoo**, y **tidyverse**, con el objetivo de facilitar el análisis, la visualización y la manipulación de datos financieros.

tidyquant se creó para simplificar el flujo de trabajo en el análisis de datos financieros en R al proporcionar una sintaxis coherente y fácil de usar. La librería permite realzar tareas comunes, como importar datos financieros, calcular indicadores técnicos, realizar análisis de rendimiento y visualizar resultados, todo dentro del marco de **tidyverse**, que permite una programación limpia y bien estructurada.

La librería **tidyquant** se creó en el año 2016 y ha ganado gran popularidad desde entonces, esto debido a su capacidad para integrar el análisis financiero en el flujo de trabajo de **tidyverse**. A consinuación se muestra un ejemplo de como utilizar la librería para importar y visualizar una serie de tiempo financiera:

```
# Instalar y cargar los paquetes necesarios
install.packages("tidyquant")
library(tidyquant)
library(ggplot2)

# Importar datos de precios de cierre de una acci n
stock_data <- tq_get("BIMBOA.MX", from = "2021-01-01", to =
    "2022-12-31")


# Visualizar la serie financiera
ggplot(stock_data, aes(x = date, y = close)) +
  geom_line(size = 1) +
  geom_smooth(method = "lm", alpha = 0.5) +
  labs(title = "Precio de cierre de BIMBOA.MX de 2021 a 2022", x =
     "Fecha", y = "Precio de cierre",
       subtitle = "Con l nea de tendencia")
```

En este ejemplo, se muestra para observar cómo importar los datos de precios de cierre de la acción "BIMBOA.MX" (Grupo Bimbo, S.A.B. de C.V.) desde el 1 de enero de 2021 hasta el 31 de diciembre del 2022. Luego, se utiliza la librería **ggplot2** para visualizar la serie financiera mendiante un diagrama de líneas[2].

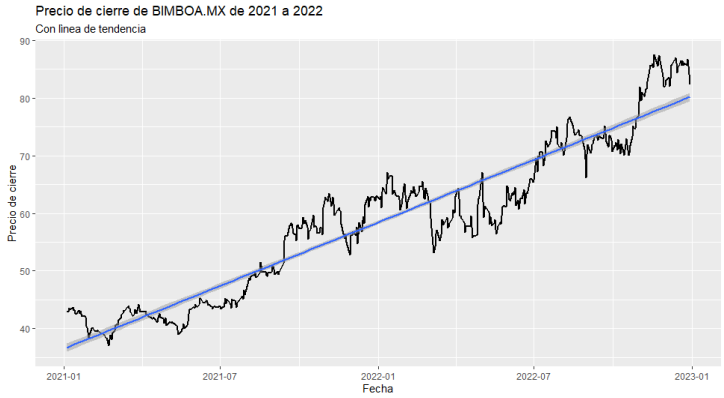

Figure 2.1: Precio de las Acciones de Grupo Bimbo

[2]Hay que considerar que este es un ejemplo básico de cómo utilizar la librería **tidyquant** para importar y visualizar una serie financiera. La librería proporciona muchas más funciones y capacidades para realizar análisis y manipulación de datos financieros dentro de R

2.2 MANIPULACIÓN DE DATOS UTILIZANDO LOS PAQUETES COMO `dplyr` Y `tidyr`

Los paquetes `dplyr` y `tidyr` son dos herramientas complementarias que se utilizan para la manipulación y transformación de datos.

Por un lado, `dplyr` es una librería ampliamente utilizada en la manipulación de datos. Proporciona una serie de verbos principales que facilitan las operaciones comunes de manipulación de datos, como seleccionar columnas, filtrar filas, ordenar datos, agregar resúmenes estadísticos y crear nuevas variables. El paquete `dplyr` se basa a trabes del concepto de tuberías (`%>%`), que permite encadenar varias operaciones de manera legible y eficiente.

Algunos de los verbos más utilizados en `dplyr`, incluyen:

- `select()`: selecciona columnas específicas del conjunto de datos.
- `filter()`: filtra filas según una o varias condiciones.
- `arrange()`: ordena los datos según una o varias columnas.
- `mutate()`: crea nuevas columnas o modifica las existentes.
- `summarize()`: calcula resúmenes estadísticos por grupos.
- `group_by()`: agrupoa los datos según una o varias columnas.

Además, el paquete incluye **operaciones de unión y combinación**, teniendo funciones para realizar operaciones de unión entre conjuntos de datos, como `inner_join()`, `left_join()`, `right_join()` y `full_join()`. Estas son funciones que permiten combinar datos en una o varias columnas clave.

Por otra parte, la librería `tidyr` proporciona funciones para transformar y reorganizar los datos, incluyendo funciones como:

- `gather()`: combinación de múltiples columnas en una columna de valores y una columna de variables.
- `spread()`: separa una columna de variables en múltiples columnas.
- `separate()`: divide una columna en múltiples columnas basadas en un separador.
- `unite()`: combina múltiples columnas en una sola columna.

Estas funciones facilitan la transformación y limpieza de datos al ajustarlos al formato de `tidy` y permite una estructura más manejable para realizar análisis posteriores.

El paquete `dplyr` es una herramienta poderosa en `R` para la manipulación y transformación de datos. Permite realizar operaciones comunes de manipulación de datos de manera eficiente y concisa.

A continuación se mostrará un uso básico y avanzado de la librería `dplyr` para el contexto de datos financieros con la librería `quantmod`:

1. Cargar las librerías a utilizar:

```
library(quantmod)
library(dplyr)
```

2. Descargar los datos financieros utilizando la librería `quantmod`:

```
# Descargar datos de Apple (AAPL) desde 2020-01-01 hasta la
    fecha actual
getSymbols("AAPL", from = "2020-01-01")

# Convertir los datos en un data frame
datos_financieros <- fortify.zoo(AAPL)
```

3. Filtr las filas basado en una condición:

```r
# Filtra los valores que se encuentren en el precio de cierre
filtrados <- datos_financieros %>%
  filter(AAPL.Close > 100)
```

4. Selecciona columnas específicas:

```r
# Selecciona solamente la columna Index y precio de acci n
   abierto
nuevos_datos <- datos_financieros %>%
  select(Index, AAPL.Open)
```

5. Ordena los datos por una columna:

```r
# Ordena los datos financieros por el precio ajustado
ordenados <- datos_financieros %>%
  arrange(AAPL.Adjusted)
```

Así mismo, como uso avanzado, la librería `dplyr` puede tener otras operaciones avanzadas para la manipulación de datos financieros.

1. Crear una nueva columna calculada:

```r
# De esta forma se consigue los rendimientos diarios de manera
   manual
datos_nuevos = datos_financieros %>%
  mutate(Daily_Return = (AAPL.Close - lag(AAPL.Close)) /
     lag(AAPL.Close))
```

2. Realiza operaciones de unión entre conjuntos de datos:

```r
# Realizar operaciones de uni n
# 1) Crear una columna 'Date'
datos_financieros <- datos_financieros %>%
  mutate(Date = time(AAPL))

# 2) Crear un segundo data frame
datos2 <- data.frame(
  Date = time(AAPL),
  Sector = rep("Tecnologia", nrow(datos_financieros)))

# 3) Unir ambos datos
datos_unidos = datos_financieros %>%
  left_join(datos_financieros, datos2, by = "Date")
```

3. Realizar operaciones de agregación y filtrado en grupos:

```r
# De esta forma se agregan y se tiene un filtrado de resumen de
   datos de cierre de precio de accion
agregados_por_mes <- datos_financieros %>%
  # Agrupamiento de los datos
  group_by(Year = year(Index), Month = month(Index)) %>%
  # Resumen por el precio de cierre de la acci n
  summarize(Average_Close = mean(AAPL.Close))
```

2.3 VISUALIZACIÓN DE DATOS FINANCIEROS UTILIZANDO ggplot2

Crear un modelo visualizado a través de la libraría `ggplot2`, se puede utilizar la función `geom_smooth()`. Esta función permite ajustar y visualizar diferentes modelos de regresión en los diagramas. A continuación se muestra un ejemplo de cómo crear un diagrama de serie financiera con `ggplot2`:

```
# Instalar y cargar los paquetes necesarios
install.packages("ggplot2")
library(ggplot2)

# Crear un conjunto de datos ficticio de precios de acciones
fecha <- seq(as.Date("2022-01-01"), as.Date("2022-12-31"), by =
    "day")
precio <- rnorm(length(fecha), mean = 100, sd = 10)
datos <- data.frame(fecha, precio)

# Crear el gr fico de serie financiera con modelo
ggplot(datos, aes(x = fecha, y = precio)) +
  geom_line(size = 0.8, color = "#76D7C4") +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Modelo de Serie Financiera volatil",
       x = "Fecha",
       y = "Precio") +
  theme_bw()
```

En este ejemplo, se ha creado un conjunto de datos ficticios del precio de acciones volátiles para un año determinado. Luego, se utiliza la función `geom_line()` para trazar la serie de precios de acciones y la función `geom_smooth()` para ajustar un modelo lineal utilizando el método de regresión, `"lm"`. El argumento `se = FALSE` se utiliza para eliminar el sombreado alrededor del modelo. También se puede explorar otros métodos disponibles que la función `geom_smooth()` ofrece para adaptar así el modelo a distintas necesidades. Teniendo entonces la **Figura 2.2**:

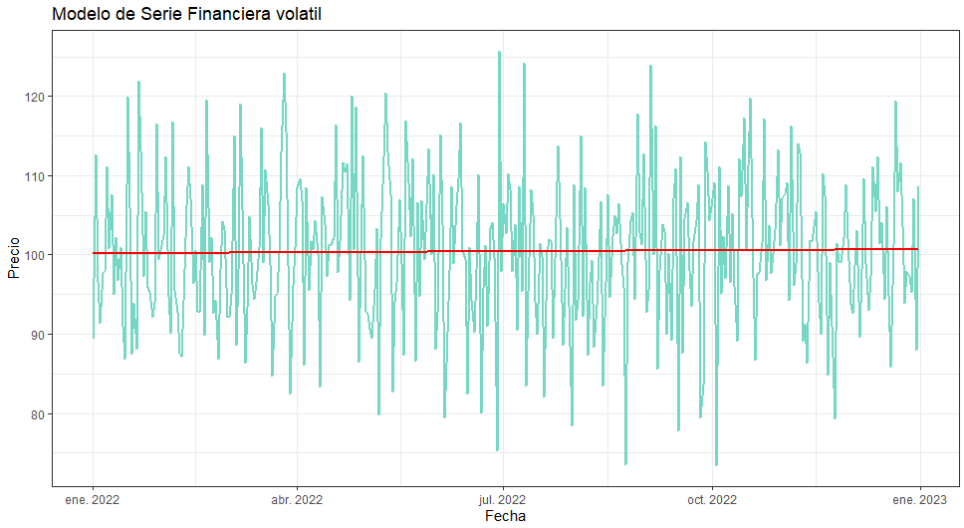

Figure 2.2: Ejemplo de diagrama de acciones de precios volátiles con `ggplot2`

Generar diagramas financieros con `ggplot2` puede requerir un poco de conocimiento adicional y ajustes específicos, pero no necesariamente se debe de considerar algo complejo o difícil de manipular. Una vez

que se le entiende a los conceptos básicos, con `ggplot2` se puede crear una amplia variedad de diagramas financieros, como los g´raficos de líneas, gráficos de velas japonesas, gráficos de barras, entre otros.

Aquí hay algunos aspectos a tener en consideración al generar diagramas financieros en R con la librería `ggplot2`:

1. **Eje X**: Debe de estar uno seguro que en el eje X esté configurado correctamente con las fechas o tiempos. Puede uno utilizar la función `as.Date()` para convertir los datos a un formato reconocible.

2. **Geometrías**: Debe de utilizarse geometrías adecuadas para representar los datos financieros. Por ejemplo, la función `geom_line()` para los diagramas lineales, `geom_bar()` para los diagramas de barras, y `geom_candlestick()` de paquetes adicionales para los gráficos de velas japonesas.

3. **Ajustes de escala**: Se puede ajustar la escala de los ejes para reflejar adecuadamente los valores financieros. Por ejemplo, se puede utilizar la función `scale_t_continuous()` para establecer los límites y las etiquetas del eje Y.

4. **Etiquetas y títulos**: Una vez terminado casi todo lo estético del diagrama, debe de asegurarse uno de agregar los títulos y etiquetas necesarias y apropiadas para identificar y representar un claro mensaje del diagrama financiero que se anda visualizando.

Si se está familiarizado con `ggplot2` y sus conceptos básicos, ajustar estos aspectos para generar gráficos financieros no debería ser demasiado difícil. Sin embargo, si uno es nuevo en `ggplot2`, puede llevar algo de tiempo familiarizarte con su sintaxis y opciones de personalización. En ese caso, explorar ejemplos, tutoriales y documentación relacionada puede ser de gran ayuda para comprender mejor las posibilidades y opciones disponibles con `ggplot2`.

A continuación se partirá de los puntos claves anteriormente mencionados para crear un diagrama de velas japonesas.

```
install.packages("ggplot2")
```

Una vez instalado y comprobado que esté abierto el paquete, se puede usar el siguiente código de ejemplo para crear un gráfico de velas japonesas con la librería `ggplot2`:

```
# Crear un conjunto de datos ficticio de precios de acciones para las velas japonesas
fecha <- seq(as.Date("2023-01-01"), as.Date("2023-05-01"), by = "day")
apertura <- rnorm(length(fecha), mean = 100, sd = 10)
cierre <- apertura + rnorm(length(fecha), mean = 0, sd = 5)
maximo <- apertura + rnorm(length(fecha), mean = 0, sd = 10)
minimo <- apertura - rnorm(length(fecha), mean = 0, sd = 10)
volumen <- rpois(length(fecha), lambda = 100)
datos <- data.frame(fecha, apertura, cierre, maximo, minimo, volumen)

# Crear el diagrama de velas japonesas utilizando ggplot2
ggplot(datos, aes(x = fecha)) +
  geom_segment(aes(x = fecha, xend = fecha, y = maximo, yend = minimo, color = ifelse(cierre > apertura, "green", "red")), size = 1) +
  geom_segment(aes(x = fecha, xend = fecha, y = apertura, yend = apertura), size = 1) +
  geom_segment(aes(x = fecha, xend = fecha, y = cierre, yend = cierre), size = 1) +
  labs(title = "Diagrama de Velas Japonesas",
       x = "Fecha",
       y = "Precio") +
  scale_color_manual(values = c("green", "red")) +
```

```
  theme_minimal() +
  theme(legend.position = "bottom", legend.title = element_blank())
```

Este ejemplo utiliza la función `geom_segment()` para dibujar las velas japonesas. Los segmentos se definen utilizando los valores de apertura, cierre, máximo y mínimo. El color de los segmentos se ajusta utilizando `ifelse()` según si el precio de cierre es mayor que el precio de apertura. También se agrega un segmento adicional para representar los precios de apertura y cierre. Teniendo así representado lo que es la **Figura 2.3**:

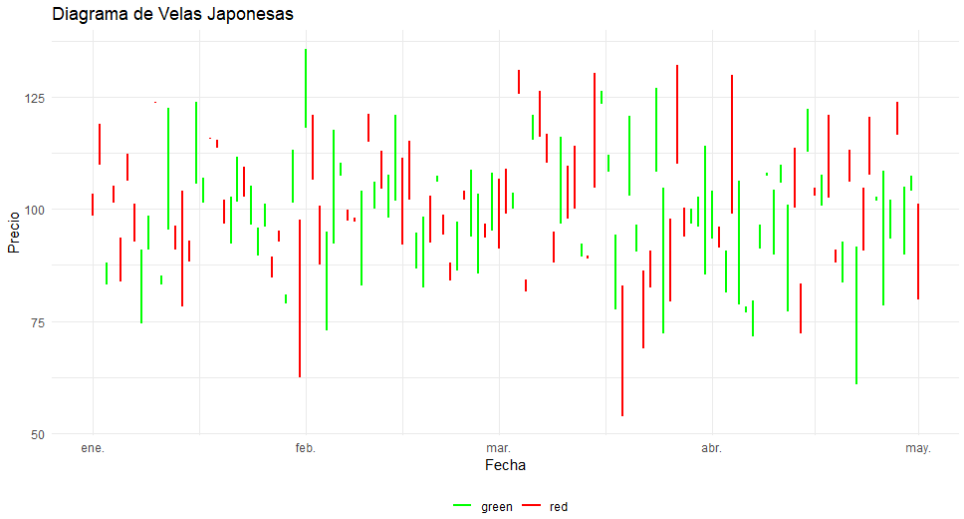

Figure 2.3: Ejemplo de velas con `ggplot2`

Para representar una serie financiera utilizando la función `geom_bar()` en `ggplot2`, se puede usar barras para mostrar los valores de la serie en diferentes períodos de tiempo. Sin embargo, se debe de tener en cuenta que la función `geom_bar()` se utiliza típicamente para representar datos categóricos, como frecuencias o conteos, y no es la opción más adecuada para una serie financiera continua. No obstante, si se desea utilizar la función `geom_bar()` para representar una serie financiera, se puede hacer utilizando datos discretizados en intervalos de tiempo específicos. A continuación se muestra un ejemplo de como crear la **Figura 2.4**:

```
# Instalar y cargar los paquetes necesarios
install.packages("ggplot2")
library(ggplot2)

# Crear un conjunto de datos ficticio de serie financiera
fecha <- seq(as.Date("2023-01-01"), as.Date("2023-12-31"), by =
    "month")
valores <- rnorm(length(fecha), mean = 1000, sd = 100)
datos <- data.frame(fecha, valores)

# Crear la gr fica de serie financiera utilizando ggplot2 y geom_bar
ggplot(datos, aes(x = fecha, y = valores)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Gr fica de Serie Financiera: Valor de una empresa
      en el mercado",
       x = "Fecha",
       y = "Valores") +
    theme_minimal()
```

En este ejemplo, se utiliza la función `geom_bar()` con el argumento `stat = "identity"` para representar los valores de la serie financiera como barras. La variable fecha se mapea en el eje X y la variable valores se mapea en el eje Y. El color de las barras se establece en `"steelblue"` utilizando el argumento `fill`. Se debe de tener en cuenta que la serie financiera se discretiza en intervalos mensuales en este caso.

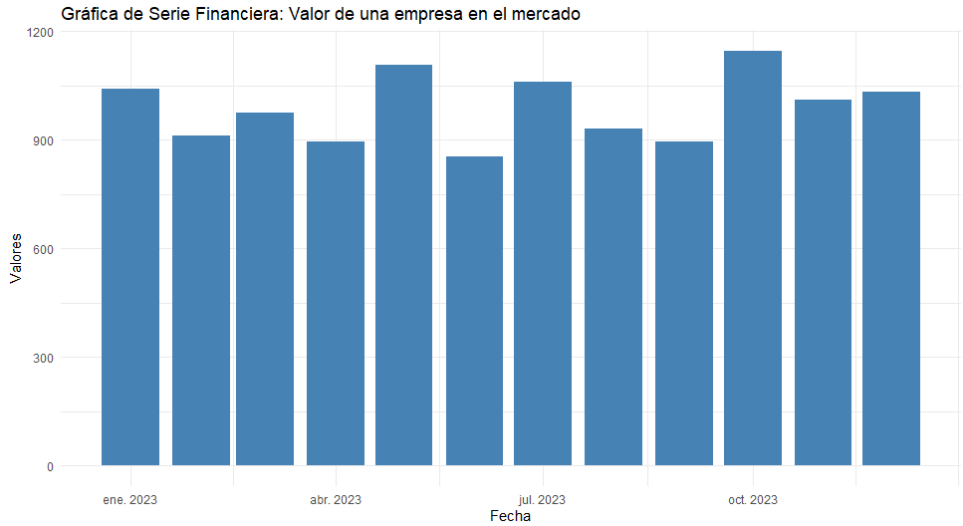

Figure 2.4: Ejemplo de barras financieras con `ggplot2`

2.3.1 La mejor manera de crear los diagramas: Creación de series de tiempo con ggplot2

Un gráfico de series temporales es una representación de datos tiempo - dependientes, esto es, es un gráfico que representa la evolución de una variable a través del tiempo. Por lo general, el par de puntos se conecta con líneas y la decisión de mostrar los puntos o no depende de gustos y del nivel de discretización.

Para crear un gráfico de series temporales en `ggplot2` de una única variable tan solo necesitas un data frame que contenga fechas y los valores correspondientes de la variable. Se debe de tener en cuenta que **la columna de fehcas debe estar en formato fecha**. Tendrás que pasar tus datos y utilizar la función `geom_line()` o `geom_point()`.

```
# install.packages("ggplot2")
library(ggplot2)

# Datos con fechas y variables. La columna 'date' (fecha) es de
    clase "Date"
df <- economics[economics$date > as.Date("2000-01-01"), ]

ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  labs(title = "Serie temporal lineal",
       x = "Fecha",
       y = "Valores") +
  theme_minimal()
```

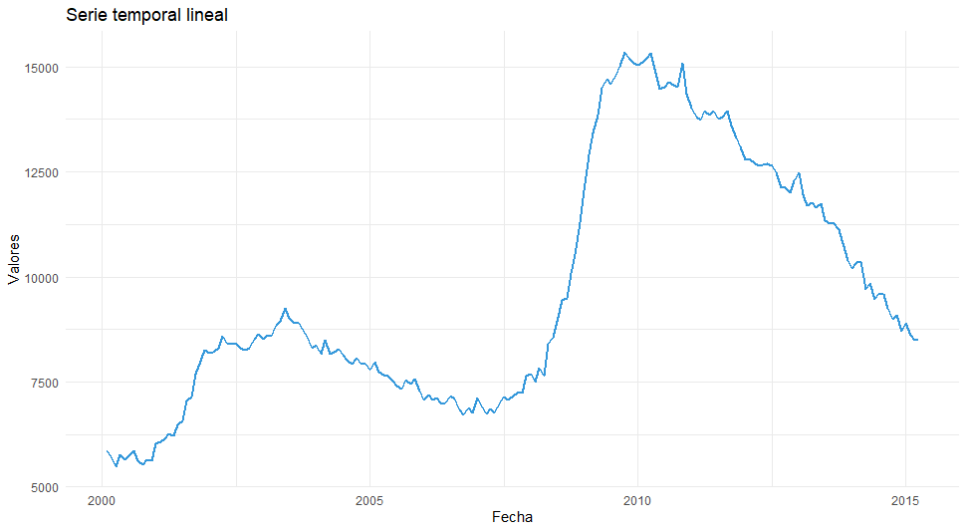

Figure 2.5: Serie temporal lineal en `ggplot2`

Tal y como se comentó antes, se pueden agregar puntos a la serie temporal utilizando la función `geom_point()`. En este ejemplo existen muchas observaciones, lo que hace que el gráfico sea menos legible con los puntos, por lo que no se mostrarán en los ejemplos siguientes:

```
ggplot(df, aes(x = date, y = unemploy)) +
  geom_line() +
  geom_point() +
  labs(title = "Serie temporal (l neas y puntos)",
       x = "Fecha",
       y = "Valores") +
  theme_minimal()
```

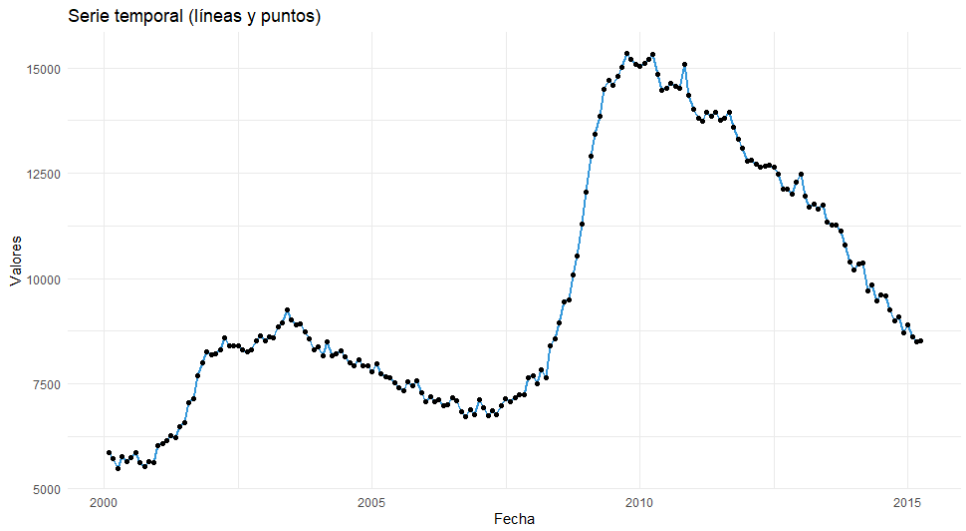

Figure 2.6: Serie temporal (líneas y puntos) `ggplot2`

En ocasiones, las series temporales se trazan con un **área sombreada**. Se puede agregar a la visualización de datos con la función `geom_area()`. Así como se muestra a continuación:

```
install.packages("ggplot2")
library(ggplot2)

# Data
df <- economics[economics$date > as.Date("2000-01-01"), ]

ggplot(df, aes(x = date, y = unemploy)) +
  geom_area(fill = "#A3E4D7", alpha = 0.5) +
  geom_line(size = 1, color = "#3498DB") +
  labs(title = " rea  de series temporales",
       x = "Fecha",
       y = "Valores") +
  theme_minimal()
```

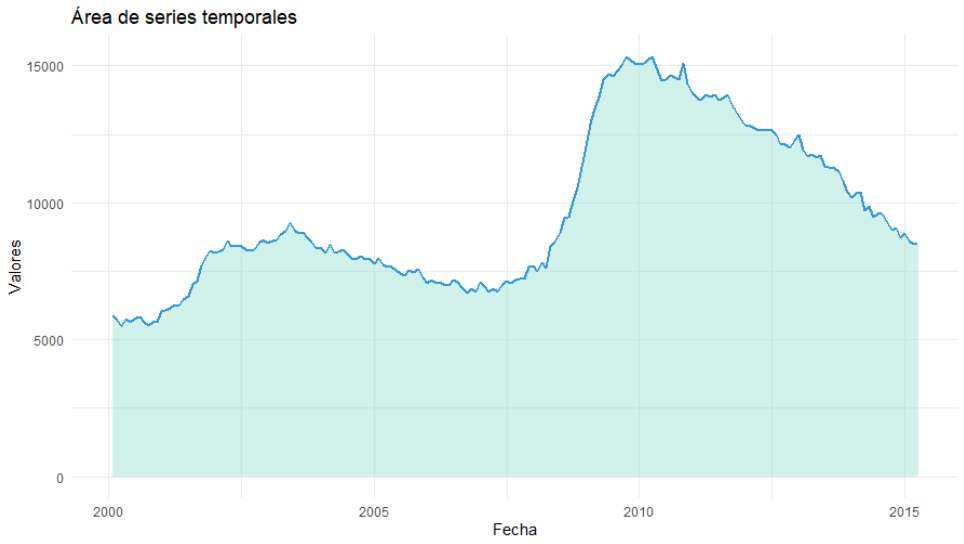

Figure 2.7: Área de series temporales

Suavizar una linea de serie e tiempo puede ser interesante y algo práctico para el análisis. De forma predeterminada se puede usar la función `stat_smooth()`.[3]

```
install.packages("ggplot2")
library(ggplot2)

# Data
df <- economics[economics$date > as.Date("2000-01-01"), ]

ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  geom_smooth(se = FALSE, color = "#7D3C98", lty = 2) +
  labs(title = "Serie tempora suavizada",
       x = "Fecha",
       y = "Valores") +
  theme_minimal()
```

[3]El color de cada línea se puede personalizar.

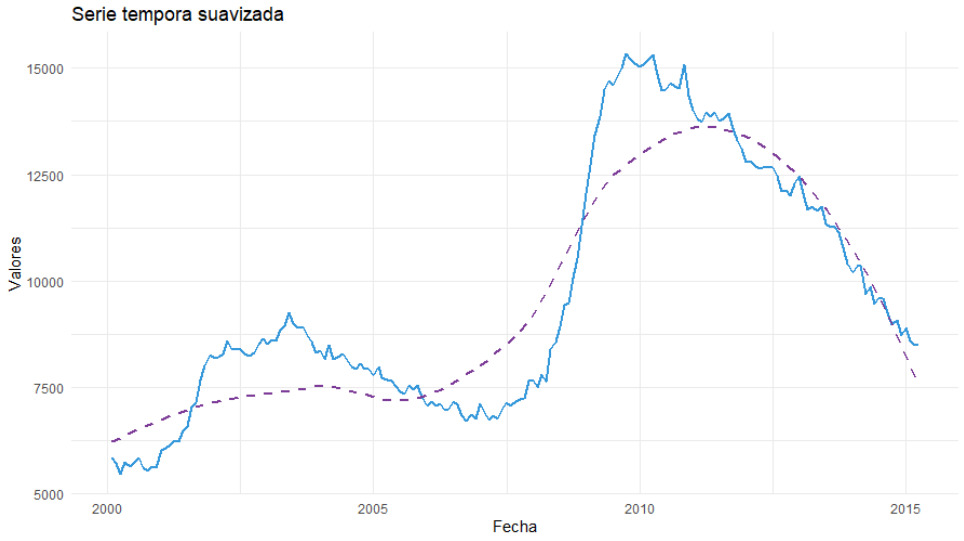

Figure 2.8: Serie temporal suavizada

Si se desea trazar dos series temporales a la vez en el mismo diagrama de serie de tiempo, se deberá de tener un marco de datos en formato largo. Para este ejemplo se usará `economics_long()` como un subconjunto del marco de datos de ejemplo de `ggplot2`. Así mismo, se usará la librería `dplyr` como para manipular los datos y mostrar doble serie.

```
# install.packages("ggplot2")
# install.packages("dplyr")
library(ggplot2)
library(dplyr)

# Data
df2 <- economics_long[economics_long$date > as.Date("2000-01-01"), ]
   %>%
  # Aqu  se filtran dos variables para la serie de tiempo que se
     grafique
  filter(variable == "pce" | variable == "unemploy")

ggplot(df2, aes(x = date, y = value, color = variable)) +
  geom_line(size = 1) +
  theme(legend.position = "bottom") +
  labs(title = "Trazar varias series de tiempo en el mismo diagrama",
     x = "Fecha",
     y = "Valores") +
  theme_minimal() +
    theme(legend.position = "bottom")
```

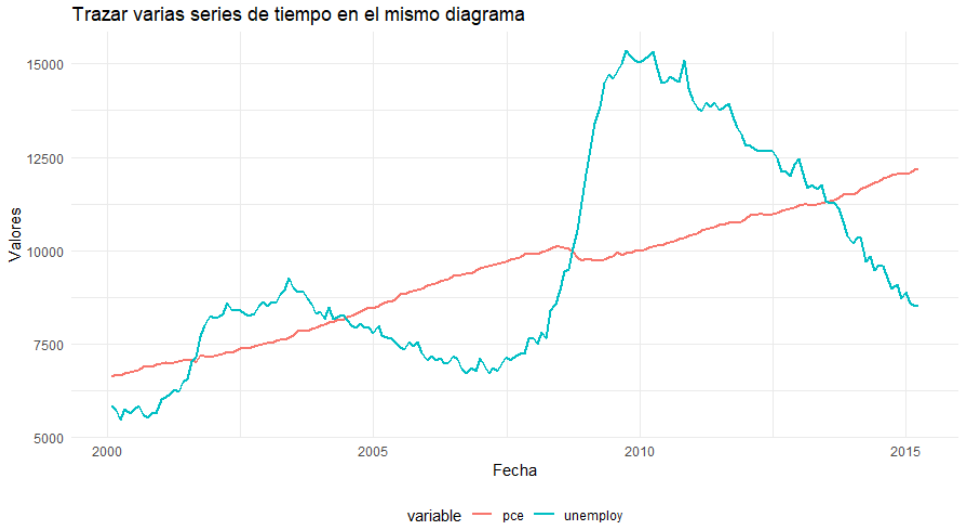

Figure 2.9: Doble serie temporal

Ahora, al hacer uso de las facetas que ofrece la librería `ggplot2`, se puede lograr dividir la serie temporar el intervalos predefinidos. El siguiente ejemplo usa y muestra la función `facet_wrap()` para lograr dividir las facetas a partir del año. Para esto también se usará la librería `lubridate`.

```
# install.packages("ggplot2")
# install.packages("lubridate")
library(ggplot2)
library(lubridate)

# Data
df <- economics[economics$date > as.Date("2000-01-01"), ]

# Nueva columna con correspondencia al a o de la variable date
df$year <- year(df$date)

ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  facet_wrap(~year, scales = "free") +
  labs(title = "Divide la serie temporal en varias parcelas por
      d a , semana, mes, a o , ...",
       x = "Fecha",
       y = "Valores") +
  theme_minimal()
```

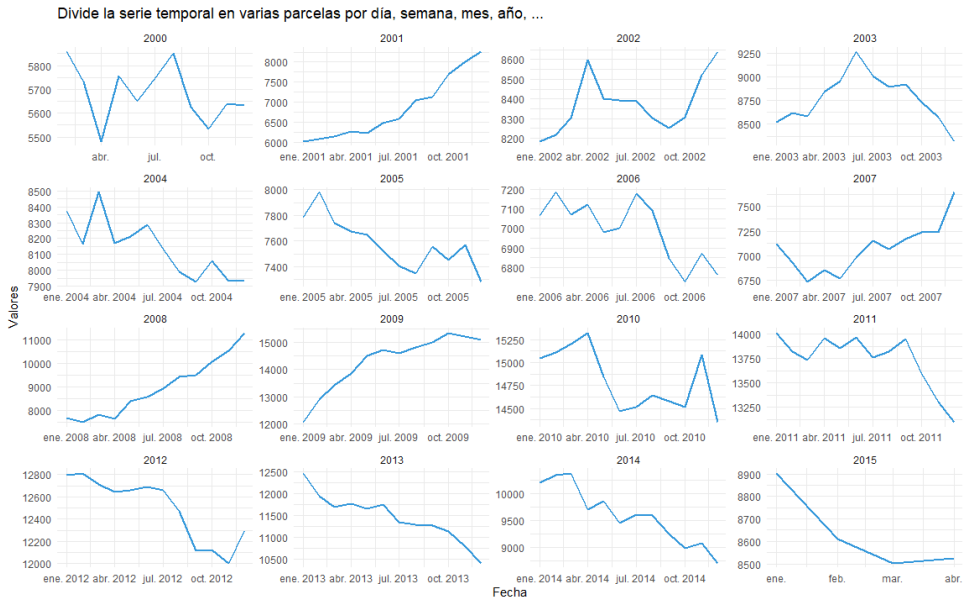

Figure 2.10: Serie dividida por facetas con `ggplot2`

Con la función `scale_x_date()` se puede personalziar la escala del eje X cuando es una fecha. Por ejemplo, se podrá transformar el formato de las fechas, los límites de la trama o el número de saltos y saltos menores del eje. En la siguiente tabla, se verá una lísta de **símbolos comunes utilizados para especificar diferentes formatos de fecha**. Hay que tener en cuenta que se pueden combinar los símbolos de la manera que se desee, como en los siguientes ejemplos.

Table 2.1: Formato de fechas de una gráfica de serie temporal

Símbolo	Significado	Ejemplo
`%Y`	4 Dígitos	2023
`%y`	2 Dígitos	23
`%B`	Nombre completo del mes	Septiembre
`%b`	Nombre abreviado del mes	Sep
`%m`	Mes como número	09
`%d`	Día del mes	05
`%H`	Hora (00-23)	14
`%I`	Hora (01-12)	14
`%M`	Minuto como número	30
`%S`	Segundo como entero	15
`%A`	Nombre completo del día de la semana	Martes
`%a`	Nombre abreviado del día de la semana	Martes

- **Formato de fecha**: El argumento `date_labels()` se puede utilizar para personalizar la fecha. En el siguiente ejemplo se establece un `scale_x_date("%Y-%m-%d")` que mostrará el año, mes y el día separados por guiones en lugar de sólo el año.
- **Límites**: Se puede establecer el límite del eje basado en fechas con el argumento `limit =`.
- **Rompe**: Se puede especificar los saltos con el argumento de la función `breaks =`, especificando las fechas que se utilizarán como saltos de eje.
- La función proporciona un argumento denominado `date_minor_breaks =` para establecer las sirrupciones menores del eje. Este argumento se omporta igual que `date_breaks()`.

```r
# install.packages("ggplot2")
# install.packages("gridExtra")
library(ggplot2)
library(gridExtra)

# Data
df <- economics[economics$date > as.Date("2000-01-01"), ]

# Ejemplo 1
p1 = ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  scale_x_date(date_labels = "%Y-%m-%d") +
  labs(title = "Ejemplo 1. Formato de fecha") +
  theme_minimal()

# Ejemplo 2
p2 = ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  scale_x_date(limit = c(as.Date("2010-01-01"),
     as.Date("2015-01-01"))) +
  labs(title = "Ejemplo 2. L mites") +
  theme_minimal()

# Ejemplo 3
p3 = ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  scale_x_date(breaks = c(as.Date("2001-01-01"),
                          as.Date("2008-01-01"),
                          as.Date("2012-01-01"))) +
  labs(title = "Ejemplo 3. Rompe") +
  theme_minimal()

# Ejemplo 4
p4 = ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  scale_x_date(date_breaks = "5 years") +
  labs(title = "Ejemplo 4. Saltos de a os") +
  theme_minimal()

# Todo junto
grid.arrange(ncol = 2, nrow = 2, p1, p2, p3, p4)
```

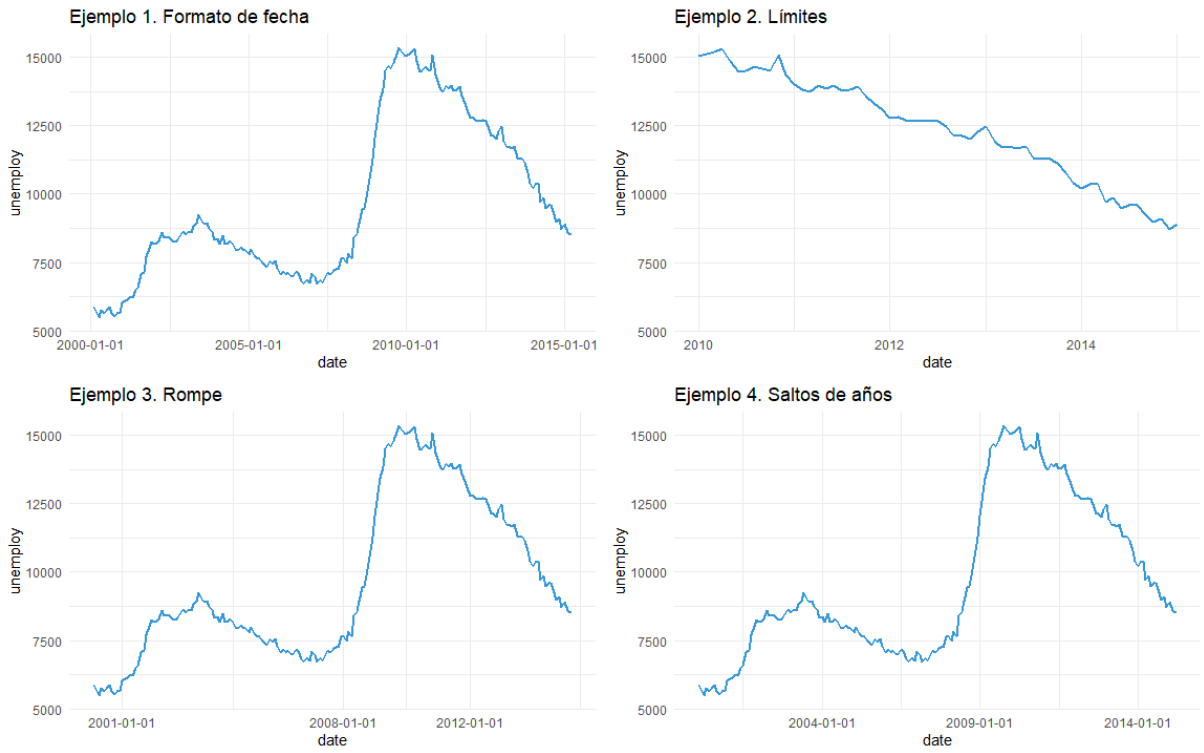

Figure 2.11: Ejemplos que proporciona los intervalos de fecha

Se pueden agregar líneas para resaltar puntos de interrupción o cambios estructurales con las funciónes `geom_vline()` o `geom_hline()` para agregar **líneas verticales u horizontales**, respectivamente.

```
# install.packages("ggplot2")
# install.packages("ggpmisc")
library(ggplot2)
library(ggpmisc)

# Data
df <- economics[economics$date > as.Date("2000-01-01"), ]

ggplot(df, aes(x = date, y = unemploy)) +
  geom_line(size = 1, color = "#3498DB") +
  geom_vline(xintercept = as.Date("2007-09-15"),
             linetype = 2, color = 2, linewidth = 1) +
  labs(title = "Resaltando cambios, picos y valles") +
  theme_minimal()
```

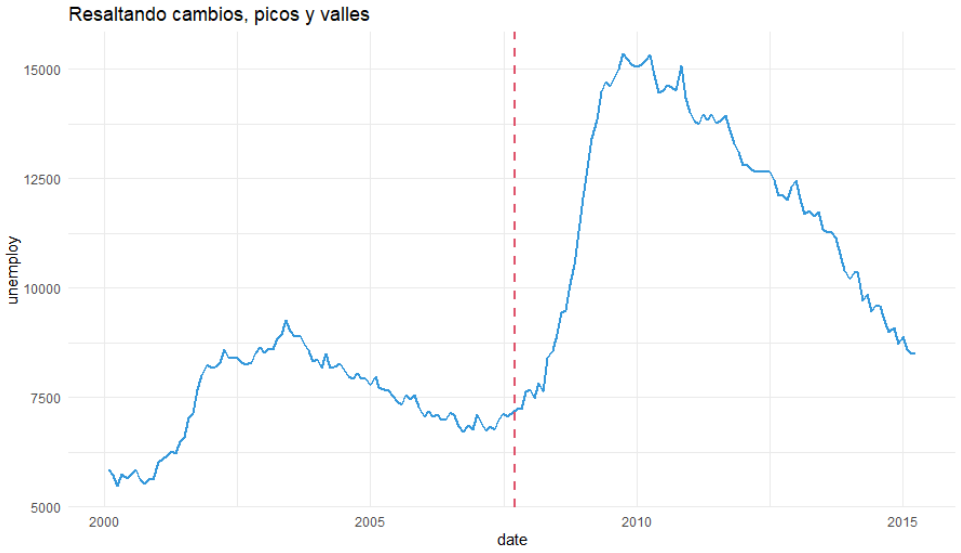

Figure 2.12: Ejemplo para resaltar cambios, picos y valles en una serie de tiempo con `ggplot2`

Por último, con la función `geom_rect()` es posible sombrear algunas áreas de la serie temporal. Esto es especialmente interesante para resaltar tendencias, recesiones, atipicidades, cambios estructurales, entre muchos otros análisis.

```
# install.packages("ggplot2")
# install.packages("ggpmisc")
library(ggplot2)
library(ggpmisc)

# Data
df <- economics[economics$date > as.Date("2000-01-01"), ]

# Shade from 2000 to 2004 and from 2010 to 2015
shade <- data.frame(x1 = c(as.Date("2000-01-01"),
    as.Date("2010-01-01")),
                    x2 = c(as.Date("2004-01-01"),
                       as.Date("2015-01-01")),
                    min = c(-Inf, -Inf), max = c(Inf, Inf))

ggplot() +
  geom_line(data = df, aes(x = date, y = unemploy), size = 1, color
      = "#3498DB") +
  geom_rect(data = shade, aes(xmin = x1, xmax = x2, ymin = min, ymax
      = max),
            fill = c("#82E0AA", "#CB4335"), alpha = 0.2) +
  labs(title = "Resaltar  reas  de sombreado") +
  theme_minimal()
```

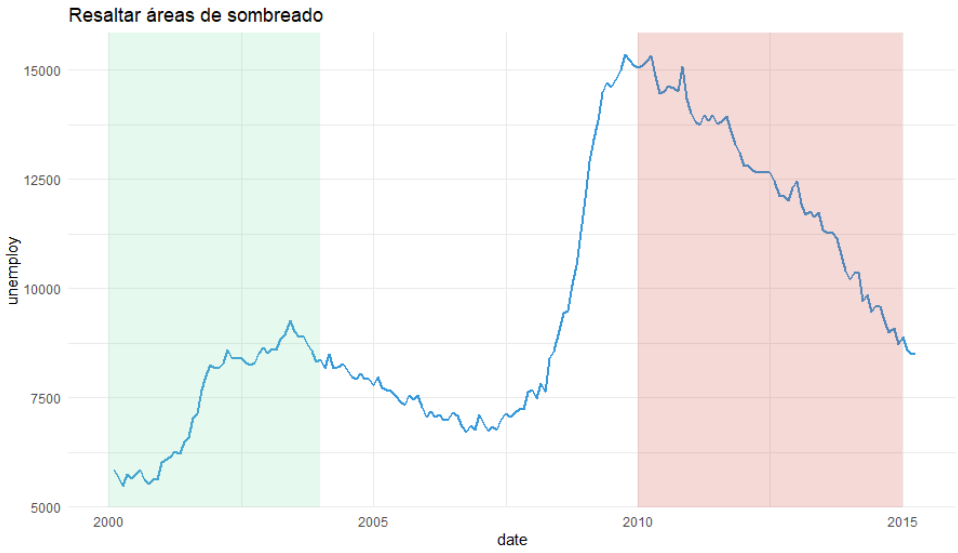

Figure 2.13: Ejemplo para resaltar áreas sombreadas `ggplot2`