

Tratamiento de datos: dplyr

Sitio: [Plataforma de Formación On line del Instituto Andaluz de
Administración Pública](#)
Curso: (I22F-PT05) Entorno de Programación R
Libro: Tratamiento de datos: dplyr

Imprimido por: ALFONSO LUIS MONTEJO RAEZ
Día: lunes, 18 de abril de 2022, 10:14

Tabla de contenidos

1. Introducción

2. Instalación

3. Trabajando con dplyr

3.1. Seleccionar variables

3.2. Filtrar variables

3.3. Ordenar

3.4. Modificar o añadir variables

3.5. Agrupar y resumir

3.6. Unir data frames

4. Pipes

4.1. Ejemplo 1

4.2. Ejemplo 2

1. Introducción



El paquete **dplyr** nos permite realizar acciones de análisis y manipulación de datos sobre data frames.

Las principales acciones que podemos hacer con este paquete son:

- Unir o dividir grandes colecciones de datos.
- Aplicar una función de resumen.
- Reagrupar variables.
- Filtrar y ordenar datos.

Este paquete está desarrollado por la empresa RStudio y utiliza la filosofía del lenguaje SQL para trabajar con los datos, además de introducir una nueva forma de programar llamada **pipe** (`%>%`).

2. Instalación

Dplyr es un paquete que hay que instalar y cargar para que funcione ya que no viene por defecto en el paquete base de R.

La instalación se puede hacer de dos formas:

- **mediante órdenes.**
- **mediante asistente.**

Mediante órdenes

Lo haríamos usando **install.packages('dplyr', dependencies=T)** y a continuación usamos la orden **library(dplyr)** para cargar el paquete y poder trabajar con él.

Mediante asistente

Como para cualquier paquete de R, el programa RStudio nos ofrece la posibilidad de instalar y cargar un paquete desde su asistente.

Este proceso ya lo hemos explicado anteriormente en la Unidad 1. Para más detalles ver:

Unidad 1 → Empezar a trabajar con RStudio → 3. Gestión de paquetes → 3.3. Instalación y 3.4. Carga

3. Trabajando con dplyr

El paquete **dplyr** nos va a permitir procesar y manipular datos contenidos en un data frame, de una forma sencilla.

Para lograr esto, dplyr nos ofrece un conjunto de órdenes y una forma de escribirlas (pipes) que trabajan sobre los datos de una manera más eficiente.

Las principales acciones que podemos hacer con este paquete son:

- Trabajar con variables (seleccionar, filtrar, ordenar, etc.)
- Unión de data frames.
- Agrupación y resumen de información.

Para poder ilustrar el uso de este paquete, vamos a usar un archivo de ejemplo, llamado tarjetas_black.RData, que contiene los datos del uso de las tarjetas black de CajaMadrid y Bankia (datos recopilados por François Delaunay).

Si tenéis curiosidad o queréis recordar de que iba este asunto aquí os dejo un [enlace](#).

Este archivo cuenta con dos data frames: uno con los movimientos de las tarjetas y otro con los usuarios de las tarjetas.

Data	
movimientos	77202 obs. of 8 variables
nombre : chr "Alberto Recarte García Andrade" "Alberto Recarte Gar..."	
fecha : POSIXct, format: "2003-01-04" "2003-01-04" ...	
hora : int 12 12 19 15 16 15 10 12 15 15 ...	
minuto : int 30 32 7 31 5 27 20 58 25 28 ...	
importe : num 38.7 14.6 95.6 49.1 13.9 ...	
comercio : chr "RCG OFICINA " "MANZANIL AREA " "REST REAL C GOLF S..."	
actividad_completa: chr "CONFECCION TEXTIL EN GENERAL" "HOTELES,MO..."	
actividad : Factor w/ 37 levels "", "AGRICULTURA",...: 28 21 27 11 1...	
usuarios	83 obs. of 3 variables
funcion : chr "concejal" "concejal" "concejal" "concejal" ...	
nombre : chr "Alberto Recarte García Andrade" "Alejandro Couceiro ...	
organizacion: chr "Partido Popular" "CEIM" "PSOE" "Izquierda Unida..."	

3.1. Seleccionar variables

Con la orden **select()** se pueden seleccionar variables de un data frame.

La forma básica de uso sería indicar el data frame del cual queremos seleccionar las variables y qué variables queremos seleccionar.

Las variables se pueden seleccionar de varias formas:

- Usando la posición de las variables dentro del data frame.
- Usando los nombres de las variables.
- Usando modificadores de selección.

Los modificadores habituales son:

- **contains('texto')**. Selecciona cualquier variable cuyo nombre contenga el texto indicado.
- **starts_with('texto')**. Selecciona cualquier variable cuyo nombre comience por el texto indicado.
- **ends_with('texto')**. Selecciona cualquier variable cuyo nombre termine por el texto indicado.
- **matches('texto')**. Selecciona cualquier variable cuyo nombre tenga una expresión regular con el texto indicado.

El resultado de la selección es otro data frame con todos los datos de las variables seleccionadas. Si queremos que el resultado se guarde como un objeto tenemos que asignar la orden.

Ejemplo

Con el fichero de ejemplo de tarjetas_black vamos a seleccionar una serie de variables usando los distintos métodos de selección.

```
> select(movimientos, c(nombre, fecha, importe))
```

	nombre	fecha	importe
1	Alberto Recarte García Andrade	2003-01-04	38.70
2	Alberto Recarte García Andrade	2003-01-04	14.60
3	Alberto Recarte García Andrade	2003-01-05	95.62

En este caso, se han seleccionado las variables nombre, fecha e importe del data frame movimientos.

```
select(movimientos, -comercio)  
select(movimientos, nombre:importe)
```

También podemos usar el negativo, para indicar que queremos todas las variables salvo esa, o los dos puntos, para seleccionar todas las variables entre las dos que indiquemos.

Otra forma de seleccionar es usar los modificadores.

```
> select(movimientos, contains('mp'))
```

	importe	actividad_completa
1	38.70	CONFECCION TEXTIL EN GENERAL
2	14.60	HOTELES, MOTELES, BALNEARIOS, CAMPINGS REST
3	95.62	RESTAURANTES RESTO
4	49.13	GASOLINERAS

Seleccionando las variables que contengan mp en su nombre.

```
> select(movimientos,starts_with('ac'))
```

	actividad_completa	actividad
1	CONFECCION TEXTIL EN GENERAL	ROPA
2	HOTELES,MOTELES,BALNEARIOS,CAMPINGS REST	HOTEL
3	RESTAURANTES RESTO	RESTAURANTE
4	GASOLINERAS	COCHE

Seleccionando las variables cuyo nombre empiece por ac.

3.2. Filtrar variables

Se pueden filtrar las filas de un data frame usando la orden **filter()**.


La forma básica de uso sería indicar el data frame a filtrar y la condición que queremos que cumplan los registros.

Las condiciones que podemos establecer se definen igual que en R. También podemos usar varias condiciones así como funciones.

El resultado de usar la orden filter() será otro data frame que, si queremos guardarlo, deberemos asignar a un objeto.

Ejemplo

Con el fichero de ejemplo de tarjetas_black vamos a filtrar todos los movimientos superiores a los 10.000 euros.



```
> filter(movimientos, importe > 10000)
```

	nombre	fecha	hora	minuto	importe
1	Ricardo Romero de Tejada y Picatoste	2007-11-26	10	59	11930.00
2	Ildefonso José Sánchez Barcoj	2006-02-15	9	13	11000.00
3	Ildefonso José Sánchez Barcoj	2009-12-31	2	40	16921.76
4	Miguel Blesa de la Parra	2006-04-05	16	51	12597.27
5	Miguel Blesa de la Parra	2006-07-20	14	50	13148.30
6	Ramón Ferraz Ricarte	2007-12-20	14	9	13549.00

También podemos filtrar uniendo condiciones.

Por ejemplo, los movimientos superiores a 5.000 euros gastados en viajes.

```
> filter(movimientos, importe > 5000 & actividad == 'VIAJE')
```

	nombre	fecha	hora	minuto	importe
1	María Carmen Cafranga Cavestany	2011-02-15	16	57	5500.00
2	Ildefonso José Sánchez Barcoj	2004-01-09	12	40	5283.34
3	Ramón Ferraz Ricarte	2003-04-04	17	56	5998.35

O filtrar las filas que pertenezcan a un conjunto.

Por ejemplo, los usuarios de las tarjetas que pertenecen al PSOE.

```
> filter(usuarios, organizacion %in% 'PSOE')
```

	funcion	nombre	organizacion
1	concejal	Ángel Eugenio Gómez del Pulgar Perales	PSOE
2	concejal	Antonio Romero Lázaro	PSOE
3	concejal	Francisco José Pérez Fernández	PSOE

O usar funciones dentro del filtrado.

Por ejemplo, buscar los movimientos realizados por personas que se llamen Rato (usando la función grepl).

```
> filter(movimientos, grepl('Rato', nombre))
```

	nombre	fecha	hora	minuto	importe
1	Rodrigo de Rato Figaredo	2010-01-31	1	25	91.74
2	Rodrigo de Rato Figaredo	2010-01-31	21	44	12.00
3	Rodrigo de Rato Figaredo	2010-02-02	15	24	13.44

3.3. Ordenar

Podemos ordenar el contenido de un data frame, en base a una o más variables, usando la orden **arrange()**.

Por defecto, todas las ordenaciones son ascendentes, pero se pueden variar usando la orden **desc()**.

La forma básica de uso sería indicar el data frame a ordenar y la variable o variables que queremos usar para la ordenación.

El resultado de la ordenación será otro data frame que, si queremos guardarlo, deberemos asignar a un objeto.

Otra forma de ordenación la podemos conseguir con la orden **top_n()**. Esta orden selecciona las n filas que tengan mayor valor en una variable.

Con esta función tenemos que indicar el data frame del que queremos obtener la información, el número de filas que queremos y la variable que se usa en la ordenación.

Ejemplo

Vamos a ordenar el data frame de usuarios por el nombre de las personas.

arrange(usuarios, nombre)

Df a ordenar

Variable a ordenar

	funcion	nombre	organizacion
1	concejal	Alberto Recarte García Andrade	Partido Popular
2	concejal	Alejandro Couceiro Ojeda	CEIM
3	concejal	Ángel Eugenio Gómez del Pulgar Perales	PSOE
4	concejal	Ángel Rizaldos González	Izquierda Unida
5	concejal	Antonio Cámara Eguinoa	Partido Popular

Si queremos hacerlo de manera descendente, tenemos que usar **desc()**.

> arrange(usuarios, desc(nombre))

	funcion	nombre	organizacion
1	concejal	Virgilio Zapatero Gómez	PSOE
2	concejal	Santiago Javier Sánchez Carlos	PSOE
3	concejal	Rubén Cruz Orive	Izquierda Unida
4	directivo	Rodrigo de Rato Figaredo	
5	concejal	Rodolfo Benito Valenciano	CCOO

También podemos usar la orden **top_n()** para obtener un ranking a partir de una variable.

Por ejemplo, obtener los cinco movimientos más altos realizados.



	nombre	fecha	hora	minuto	importe		
1	Ildefonso José Sánchez Barcoj	2009-12-31	2	40	16921.76	VIAJES	ECI
2	Miguel Blesa de la Parra	2006-04-05	16	51	12597.27	VIAJES	ECI
3	Miguel Blesa de la Parra	2006-07-20	14	50	13148.30	VIAJES	ECI
4	Ramón Ferraz Ricarte	2007-12-20	14	9	13549.00		
5	Matías Amat Roca	2006-12-27	17	6	15000.00	TALLERES DE ARTE	

La orden selecciona los cinco gastos más altos, pero los muestra según los ha ido encontrado dentro del data frame por eso aparecen desordenados.

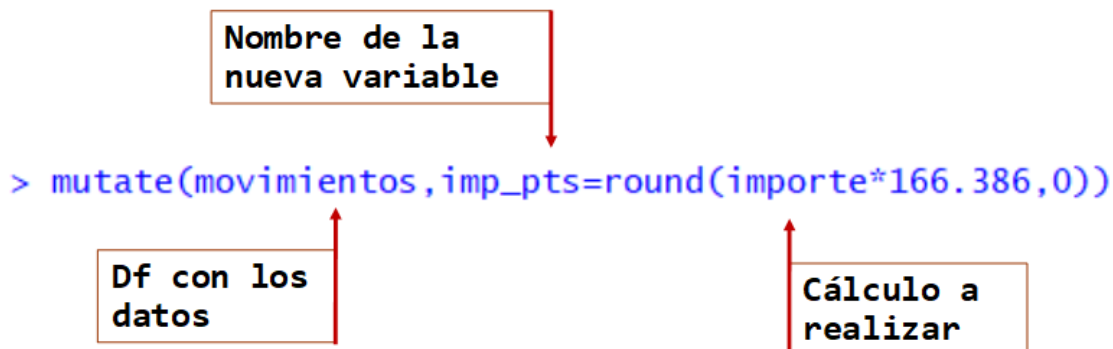
3.4. Modificar o añadir variables

Podemos modificar variables existentes o añadir nuevas variables a partir de cálculos usando la orden `mutate()`.

Ejemplo

Dentro del data frame de movimientos, vamos a crear una nueva variable donde vamos a almacenar los importes convertidos en pesetas y vamos a modificar una variable existente pasando los valores de la variable nombre a mayúsculas.

En primer lugar creamos la variable convirtiendo los importes en euros a pesetas.



	nombre				fecha	importe	imp_pts
1	Alberto	Recarte	García	Andrade	2003-01-04	38.70	6439
2	Alberto	Recarte	García	Andrade	2003-01-04	14.60	2429
3	Alberto	Recarte	García	Andrade	2003-01-05	95.62	15910
4	Alberto	Recarte	García	Andrade	2003-01-08	49.13	8175
5	Alberto	Recarte	García	Andrade	2003-01-08	13.94	2319
6	Alberto	Recarte	García	Andrade	2003-01-09	80.00	13311
7	Alberto	Recarte	García	Andrade	2003-01-11	53.37	8880
8	Alberto	Recarte	García	Andrade	2003-01-11	42.00	6988
9	Alberto	Recarte	García	Andrade	2003-01-15	263.30	43809
10	Alberto	Recarte	García	Andrade	2003-01-21	2.10	349

Después modificamos una variable existente, en este caso pasamos a mayúsculas la variable nombre.

```
> mutate(movimientos,nombre=toupper(nombre))
```

	nombre				fecha	hora	minuto	importe
1	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-04	12	30	38.70
2	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-04	12	32	14.60
3	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-05	19	7	95.62
4	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-08	15	31	49.13
5	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-08	16	5	13.94
6	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-09	15	27	80.00
7	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-11	10	20	53.37
8	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-11	12	58	42.00
9	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-15	15	25	263.30
10	ALBERTO	RECARTE	GARCÍA	ANDRADE	2003-01-21	15	28	2.10

En estos casos hay que tener cuidado con el nombre de la variable que queremos usar. Si usamos el nombre de una variable que ya existe en el data frame, la machacamos y perdemos su contenido anterior.

Como siempre, si el resultado de las órdenes no lo asignamos a un objeto los cambios no se guardan y solo mostramos las operaciones por pantalla.

3.5. Agrupar y resumir

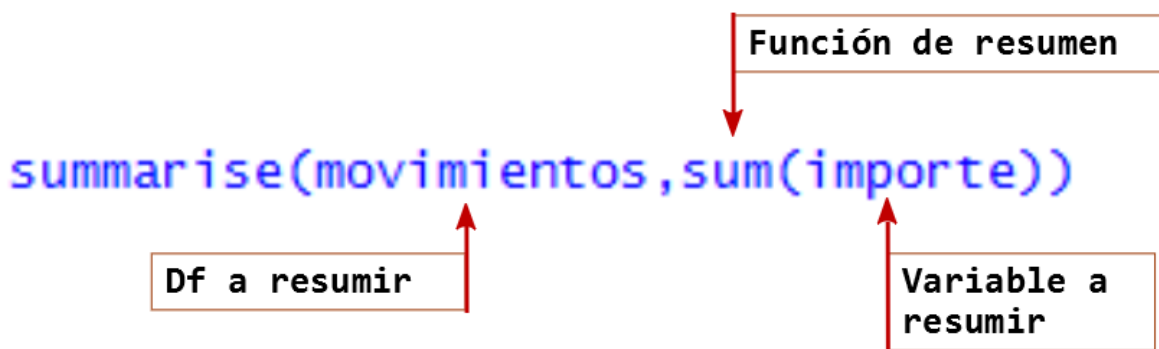
Otras de las opciones que tenemos con este paquete son agrupar información por variables y resumir datos mediante funciones.

Las órdenes básicas para esto son **group_by()** y **summarise()**.

Son dos funciones que suelen ir conjuntamente, aunque se pueden usar por separado. Con la orden **group_by()** seleccionamos las variables de agrupación (normalmente categóricas) y con la orden **summarise()** generamos resúmenes estadísticos de diferentes variables en el data frame por ejemplo, usando una suma, una media, etc.

Ejemplo

Con el data frame de las tarjetas black vamos a calcular el total de dinero gastado y el gasto máximo realizado por usuario.

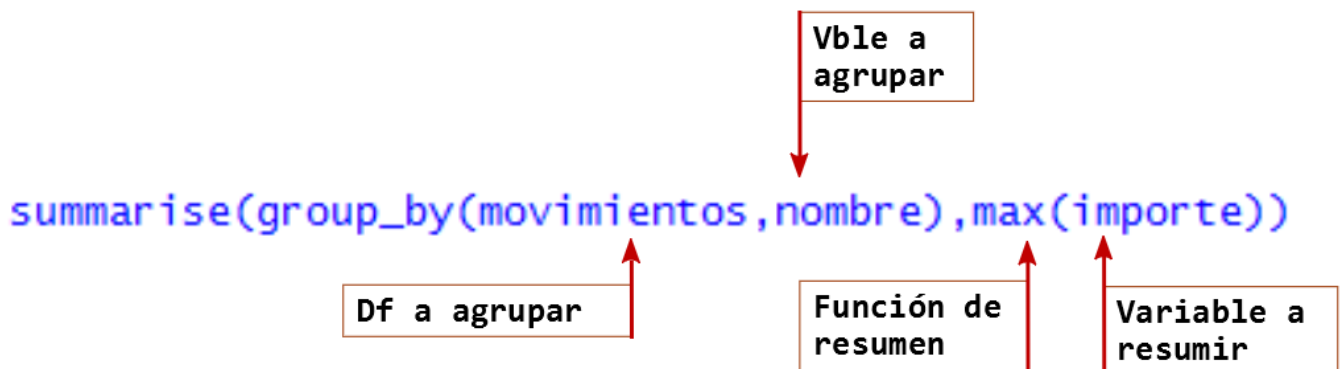


En este caso solo usamos la orden **summarise()**, ya que afecta a todos los datos y no tenemos que agrupar, indicando el data frame que contiene los datos (movimientos), la función que queremos usar (sum) y cual es la variable que queremos resumir (importe).

```
sum(importe)
11806773
```

En total se han gastado 11.806.733 euros

Para calcular el gasto medio por usuario ya tenemos que usar la función de agrupación junto con la de resumen.



Aquí usamos las dos funciones anidadas (una dentro de otra) indicando el data frame (movimientos), la variable que queremos agrupar (nombre), la función de resumen (max) y sobre que variable queremos que actúe el resumen (importe).

	nombre	<code>`max(importe)`</code>
	<code><chr></code>	<code><dbl></code>
1	Alberto Recarte García Andrade	3509.
2	Alejandro Couceiro Ojeda	1150
3	Ángel Eugenio Gómez del Pulgar Perales	4906
4	Angel Rizaldos González	843
5	Antonio Cámara Eguinoa	2742

Como resultado tenemos la tabla con los movimientos más altos realizados por cada persona.

Si queremos guardar el resultado, tenemos que asignar la orden a un objeto. El resultado es un formato similar a un data frame con dos variables (nombre y max(importe)).

Como véis, el nombre de la segunda variable (la que hemos usado para la agrupación), toma automáticamente el nombre de la función que hemos usado. Para cambiar esto, podemos darle nombre a la variable que se generan con el resumen.

Por ejemplo, vamos a llamar max_persona a esa variable de agrupación.

Se renombra la variable

↓

```
summarise(group_by(movimientos,nombre),max_persona=max(importe))->maximo
```

Se crea un nuevo df llamado maximo

↑

	nombre	max_persona
1	Alberto Recarte García Andrade	3509.20
2	Alejandro Couceiro Ojeda	1150.00
3	Ángel Eugenio Gómez del Pulgar Perales	4906.00
4	Angel Rizaldos González	843.00
5	Antonio Cámara Eguinoa	2742.00

También podemos resumir de varias formas al mismo tiempo. Solo tenemos, dentro de la orden summarise(), las funciones que queramos separadas por comas.

Incluimos dos variables

↓

```
summarise(group_by(movimientos,nombre),max_persona=max(importe),gasto_medio=mean(importe))
```

Por ejemplo, aquí tenemos el importe máximo de los movimientos realizados por una persona y su gasto medio.

3.6. Unir data frames

Con el paquete **dplyr** también podemos unir dos data frames (A y B) usando una variable común entre ellos usando un concepto similar a la unión en lenguaje SQL.

El resultado de la unión es otro data frame con las columnas unidas a partir de los data frames A y B.

Los órdenes para realizar los distintos tipos de unión son:

- **left_join()**: Unión de B común hacia A.
- **right_join()**: Unión de A común hacia B.
- **inner_join()**: Solo mantiene lo común en A y B.
- **full_join()**: Unión completa de A y B.
- **anti_join()**: Lo que está en un data frame y no en otro.

En todas las órdenes se usa el parámetro **by** que sirve para indicar cual es el campo (o campos) común entre los dos data frames.

Ejemplo

Vamos a ver un ejemplo con las distintas formas de unión usando un nuevo conjunto de datos llamado **ventas**.

Este archivo contiene dos data frames: uno con el nombre de los clientes y otro con las ventas realizadas a esos clientes. Estos data frames tienen en común el campo **id** que sirve de indentificador del cliente.

clientes x			
Filter			
	id		nombre
1	1		Carlos
2	2		Sara
3	3		Raquel
4	4		Miguel

Ambos comparten
el campo **id**

ventas x			
Filter			
	id	fecha	total
1	1	Ene	72.54347
2	2	Ene	30.71759
3	3	Ene	53.96646
4	2	Feb	70.67072
5	3	Feb	70.56104
6	1	Mar	69.99067
7	2	Mar	54.30900
8	3	Mar	87.98441
9	3	Abr	76.62748
10	5	Abr	28.19212

left_join

```
> left_join(clientes,ventas,by='id')
  id nombre fecha  total
1  1 Carlos  Ene 72.54347
2  1 Carlos  Mar 69.99067
3  2 Sara   Ene 30.71759
4  2 Sara   Feb 70.67072
5  2 Sara   Mar 54.30900
6  3 Raquel Ene 53.96646
7  3 Raquel Feb 70.56104
8  3 Raquel Mar 87.98441
9  3 Raquel Abr 76.62748
10 4 Miguel <NA>      NA
```

Con la orden **left_join()**, tenemos como resultado una tabla con los datos de clientes y ventas unidas.

En el caso de Miguel, como no ha realizado ninguna compra, aparece con el valor NA en los datos correspondientes a las ventas, ya que está solo en el data frame de clientes y no en el de ventas.

right_join

```
> right_join(clientes,ventas,by='id')
  id nombre fecha    total
1  1 Carlos  Ene  72.54347
2  2 Sara   Ene  30.71759
3  3 Raquel Ene  53.96646
4  2 Sara   Feb  70.67072
5  3 Raquel Feb  70.56104
6  1 Carlos Mar  69.99067
7  2 Sara   Mar  54.30900
8  3 Raquel Mar  87.98441
9  3 Raquel Abr  76.62748
10 5 <NA>    Abr  28.19212
```

Con la orden **right_join()**, tenemos como resultado una tabla con los datos de clientes y ventas unidas, pero incorporando los datos de clientes.

En este caso aparece un id 5, que no corresponde a ningún cliente, por lo que sus datos quedan en blanco (NA) en la parte de clientes y rellenos en la parte de ventas porque si ha realizado compras.

inner_join

```
> inner_join(clientes,ventas,by='id')
  id nombre fecha    total
1  1 Carlos  Ene  72.54347
2  1 Carlos  Mar  69.99067
3  2 Sara   Ene  30.71759
4  2 Sara   Feb  70.67072
5  2 Sara   Mar  54.30900
6  3 Raquel Ene  53.96646
7  3 Raquel Feb  70.56104
8  3 Raquel Mar  87.98441
9  3 Raquel Abr  76.62748
```

Con la orden **inner_join()**, tenemos como resultado una tabla con los datos de clientes y ventas que aparecen en los dos data frames simultáneamente.

En este caso no aparece la compra hecha por el id 5 (ya que no existe en la tabla de clientes) ni el cliente Miguel (ya que no ha comprado nada). Solo aparece lo que está en los dos data frames al mismo tiempo.

full_join

```
> full_join(clientes,ventas,by='id')
  id nombre fecha    total
1  1 Carlos  Ene  72.54347
2  1 Carlos  Mar  69.99067
3  2 Sara   Ene  30.71759
4  2 Sara   Feb  70.67072
5  2 Sara   Mar  54.30900
6  3 Raquel Ene  53.96646
7  3 Raquel Feb  70.56104
8  3 Raquel Mar  87.98441
9  3 Raquel Abr  76.62748
10 4 Miguel <NA>    NA
11 5 <NA>    Abr  28.19212
```

Con la orden **full_join()**, tenemos como resultado una tabla con la unión total de los datos de clientes y ventas.

En este caso aparecen todos los datos unidos aunque se encuentren solo en uno de los dos data frames. Los datos que falten en alguno de los dos, aparecerán como NA.

Así, no aparece datos de ventas a Miguel, ni datos de clientes para el id 5.

anti_join

```
> anti_join(clientes,ventas,by='id')
  id nombre
1  4 Miguel
> anti_join(ventas,clientes,by='id')
  id fecha    total
1  5 Abr  28.19212
```

Con la orden **anti_join()**, tenemos como resultado una tabla con lo que falta en cada uno de los data frames.

En el primer caso, tenemos los datos que están en clientes pero no aparecen en ventas. Miguel es cliente pero no ha hecho compras.

En el segundo caso, tenemos los datos que están en ventas pero no aparecen en clientes. La venta realizada por el id 5, que no está dado de alta como cliente.

4. Pipes

Una de las ventajas del paquete **dplyr** es introducir un modo de encadenar acciones de una forma muy sencilla.

La filosofía es que cada función aplicada le pasa el resultado a siguiente mediante llamados **pipes o tuberías** (%>%).

De esta forma, escribir un código complejo que requiera de varias funciones, se simplifica ya que van encadenando acciones mediante los pipes y cada acción pasa su resultado a la siguiente para que continúe con él.

Ejemplo 1

Queremos calcular el logaritmo de 2.4 y luego, con su resultado, calcular el coseno y posteriormente la raíz cuadrada.

Esto podemos hacerlo anidando las funciones una dentro de otra, colocando la primera en la parte más interior de las funciones. Es un poco engorroso y poco legible.

```
> sqrt(cos(log(2.4)))  
[1] 0.800398
```

Sin embargo, si usamos el operador pipe (%>%) vamos escribiendo todas las funciones de una manera más lógica y legible.

```
> log(2.4)%>%cos()%>%sqrt()  
[1] 0.800398
```

El resultado del logaritmo se pasa al siguiente paso, que es el coseno, y así sucesivamente hasta aplicar todas las funciones.

Ejemplo 2

Con el data frame de las tarjetas black vamos a obtener los cinco gastos más importantes ordenados y mostrando la persona, el importe y la actividad donde se ha realizado el gasto.

Con el uso de los pipes podemos encadenar varias acciones de la siguiente manera.

```
select(movimientos,c(nombre,importe,actividad))%>%  
  arrange(desc(importe))%>%  
  top_n(5,importe)
```

Con estas órdenes seguimos el siguiente esquema:

1. Seleccionamos el data frame y las variables que queremos mostrar.
2. Ordenamos de manera descendente el importe.
3. Nos quedamos con las cinco primeras filas de la tabla ordenada.

El resultado sería:

	nombre	importe	actividad
1	Ildefonso José Sánchez Barcoj	16921.76	COMPRA BIENES
2	Matías Amat Roca	15000.00	HOGAR
3	Ramón Ferraz Ricarte	13549.00	BANCO
4	Miguel Blesa de la Parra	13148.30	COMPRA BIENES
5	Miguel Blesa de la Parra	12597.27	COMPRA BIENES

4.1. Ejemplo 1

A partir de los datos de las tarjetas black, obtener un listado con las diez personas que más dinero han gastado en total.

```
select(movimientos,c(nombre,importe))%>%  
  group_by(nombre)%>%  
  summarise(importe=sum(importe))%>%  
  arrange(desc(importe))%>%  
  top_n(10,importe)
```

Con estas órdenes seguimos el siguiente esquema:

1. Seleccionamos el data frame y las variables que queremos mostrar.
2. Agrupamos la información por el nombre de cada persona.
3. Resumimos la información como suma de todos los importes por persona, usando la función `sum()`.
4. Ordenamos de manera descendente por el importe total calculado.
5. Nos quedamos con las diez primeras filas de la tabla ordenada.

El resultado sería:

	nombre <chr>	importe <dbl>
1	Ildefonso José Sánchez Barcoj	572187.
2	José Antonio Moral Santín	447770.
3	Ricardo Morado Iglesias	443999.
4	Matías Amat Roca	429272.
5	Miguel Blesa de la Parra	423068.
6	Ramón Ferraz Ricarte	389450.
7	Mariano Pérez Claver	354171.
8	Enrique de la Torre Martínez	304818.
9	Juan Manuel Astorqui Portera	287121.
10	Maria Mercedes de la Merced Monge	284404.

4.2. Ejemplo 2

A partir de los datos de las tarjetas black, obtener un listado con el gasto medio por organización a la que pertenecen las personas.

```
left_join(movimientos, usuarios, by = 'nombre') %>%
  select(organizacion, importe) %>%
  group_by(organizacion) %>%
  summarise(media = mean(importe))
```

Con estas órdenes seguimos el siguiente esquema:

1. Unimos los data frames movimientos y usuarios usando el nombre de las personas como campo de unión. El dato de la organización de las personas está en el data frame usuario.
2. Seleccionamos las variables organización e importe, que son las que nos interesan.
3. Agrupamos la información por organización.
4. Resumimos la información como la media de todos los importes realizados por persona de cada organización, usando la función mean().

El resultado sería:

	organizacion	media
	<i><chr></i>	<i><dbl></i>
1	""	295.
2	"CCOO"	126.
3	"CEIM"	179.
4	"CEOE"	1375.
5	"Comisión de Control"	92.0
6	"Conf. de Cuadros"	129.
7	"Izquierda Unida"	145.
8	"Partido Popular"	114.
9	"Patronal (Unipyme)"	162.
10	"PSOE"	97.2
11	"UGT"	106.
12	NA	227.