thePower

Uso de librería dplyr para la manipulación de datos

Funciones de `dplyr`:

1. `filter()`:
   - **Para qué sirve:** Filtra filas del dataframe basándose en condiciones específicas.
   - **Cuándo utilizar:** Cuando deseas extraer un subconjunto de datos que cumpla con ciertas condiciones.
   - **Ejemplo:**

```
library(dplyr)

# Filtrar filas donde la variable 'edad' es mayor que 25
df_filtrado <- df %>% filter(edad > 25)
```

2. `select()`:
   - **Para qué sirve:** Selecciona columnas específicas del dataframe.
   - **Cuándo utilizar:** Cuando solo necesitas trabajar con un subconjunto de variables.
   - **Ejemplo:**

```
# Seleccionar las columnas 'nombre' y 'puntuacion'
df_seleccionado <- df %>% select(nombre, puntuacion)
```

3. `mutate()`:
   - **Para qué sirve:** Agrega nuevas columnas al dataframe o modifica las ya existentes.
   - **Cuándo utilizar:** Cuando necesitas realizar cálculos o transformaciones en las variables existentes o crear nuevas variables.
   - **Ejemplo:**

```
# Crear una nueva columna 'puntuacion_doble'
df_modificado <- df %>% mutate(puntuacion_doble = puntuacion * 2)
```

4. `arrange()`:
   - **Para qué sirve:** Ordena las filas del dataframe basándose en el valor de una o varias columnas.
   - **Cuándo utilizar:** Cuando necesitas ver o analizar los datos en un orden específico.
   - **Ejemplo:**

```
# Ordenar el dataframe por la variable 'edad' de forma descendente
df_ordenado <- df %>% arrange(desc(edad))
```

5. `group_by()`:
   - **Para qué sirve:** Agrupa las filas del dataframe basándose en los valores de una o varias columnas.
   - **Cuándo utilizar:** Cuando necesitas realizar cálculos o resúmenes por grupos.
   - **Ejemplo:**

```
# Agrupar por la variable 'genero'
df_agrupado <- df %>% group_by(genero)
```

6. `summarize()`:
   - **Para qué sirve:** Calcula resúmenes y estadísticas agregadas para cada grupo.
   - **Cuándo utilizar:** Después de agrupar datos, para obtener estadísticas resumidas para cada grupo.
   - **Ejemplo:**

```
# Calcular la media y la desviación estándar de la variable 'puntuacion' por g
rupo
resumen_grupo <- df_agrupado %>% summarize(media_puntuacion = mean(puntuacio
```

```
n),
                                                sd_puntuacion = sd(puntuacion))
```

Operador Tubería (`%>%` o pipe):

El operador tubería simplifica el código al permitir que el resultado de una expresión se utilice como entrada para la siguiente. Facilita la lectura del código y reduce la necesidad de crear variables intermedias.

Ejemplo de Operador Tubería:

```
# Sin usar tubería
df_filtrado <- filter(select(df, nombre, edad), edad > 25)

# Usando tubería
df_filtrado <- df %>% select(nombre, edad) %>% filter(edad > 25)
```

Algunos ejemplos de casos de uso de negocio en los que la librería `dplyr` sería especialmente útil para realizar manipulaciones y análisis de datos en R:

1. **Análisis de Ventas:**
   - **Escenario:** Tienes un conjunto de datos que contiene información de ventas, incluyendo productos, fechas y cantidades vendidas.
   - **Uso de `dplyr`:** Puedes utilizar `filter` para seleccionar datos de un período específico, `group_by` para agrupar por productos, y `summarize` para calcular la cantidad total de productos vendidos por categoría.

```
df_ventas_anual <- df_ventas %>%
  filter(year(fecha) == 2022) %>%
  group_by(categoria_producto) %>%
  summarize(total_ventas = sum(cantidad))
```

2. **Segmentación de Clientes:**
   - **Escenario:** Tienes datos de clientes que incluyen información demográfica, historial de compras y preferencias.
   - **Uso de `dplyr`:** Puedes utilizar `mutate` para crear nuevas variables, `filter` para segmentar clientes según ciertos criterios y `group_by` con `summarize` para obtener estadísticas resumidas por segmento.

```
df_clientes_segmentados <- df_clientes %>%
  mutate(edad_grupo = case_when(edad < 25 ~ 'Joven',
                                edad >= 25 & edad < 40 ~ 'Adulto',
                                edad >= 40 ~ 'Mayor')) %>%
  group_by(edad_grupo) %>%
  summarize(promedio_gasto = mean(gasto_total))
```

3. **Análisis de Recursos Humanos:**
   - **Escenario:** Tienes datos de recursos humanos que incluyen información sobre empleados, salarios y evaluaciones de desempeño.
   - **Uso de `dplyr`:** Puedes utilizar `arrange` para ordenar los empleados por salario, `filter` para seleccionar empleados de un departamento específico y `mutate` para calcular ratios de desempeño.

```
df_rrhh_analisis <- df_rrhh %>%
  arrange(desc(salario)) %>%
  filter(departamento == 'Ventas') %>%
  mutate(ratio_desempeno = evaluacion_desempeno / salario)
```

4. **Análisis de Marketing:**
   - **Escenario:** Tienes datos de campañas de marketing que incluyen información sobre canales, conversiones y costos.
   - **Uso de `dplyr`:** Puedes utilizar `group_by` para agrupar por canal, `summarize` para calcular el costo promedio por conversión y `filter` para seleccionar campañas específicas.

```
df_marketing_analisis <- df_marketing %>%
  group_by(canal) %>%
  summarize(costo_promedio_por_conversion = sum(costo) / sum(conversiones)) %>%
  filter(costo_promedio_por_conversion < 5)
```

Estos ejemplos muestran cómo `dplyr` puede ser empleado en una variedad de situaciones de negocio para realizar análisis y manipulación de datos de manera eficiente y legible. La sintaxis clara y concisa de `dplyr` facilita la implementación de estos análisis, mejorando la productividad y la comprensión del código.

Conclusión:

La librería `dplyr` es esencial para realizar manipulación de datos eficiente en R. Sus funciones proporcionan un flujo de trabajo claro y conciso, simplificando tareas comunes en análisis de datos. El operador tubería mejora aún más la legibilidad del código. En un entorno de análisis de datos real, `dplyr` se convierte en una herramienta indispensable para explorar, transformar y resumir datos de manera eficiente y efectiva. Su sintaxis intuitiva y consistente facilita la adopción y mejora la productividad en el análisis de datos en R.