Chapter 5 *dplyr* Package

In this course, data transformations are primarily performed using the `dplyr` package (pronounced DEE ply er). This package makes data manipulation easier and more intuitive (for most). `dplyr` is built around the five main "verbs" shown below that make up a majority of data manipulation. However, there are other functions that dplyr uses to also help with data manipulation.

- `select` is used to subset columns
- `filter` is used to subset rows
- `mutate` is used to add new columns based on calculations (usually with other columns)
- `summarise` is use to perform summary calculations (e.g., mean, max, etc.) on data set
- `group_by` is used to group rows of a data frame with the same value in specified columns

In addition, `dplyr` uses the pipe, `%>%`, to string together a series of functions. Think of functions strung together as upstream and downstream functions. The function to the left of `%>%` is the upstream function, while the function to the right is the downstream function.

By default, the downstream function assumes the value coming from the upstream function is the first argument in its function. Therefore, the first argument can be omitted. If the downstream function needs to use the value from the upstream function assigned to a different argument, a `.` is simply put in the position of that argument

---

5.1 The *dplyr* workflow

The `dplyr` package is designed to make data manipulation clear and readable.

```
library(dplyr)
```

A typical `dplyr` workflow:

1. Start with a data frame
2. Apply a sequence of transformation verbs
3. Save or display the result

5.2 Selecting variables

Use `select()` to keep only the variables you need. This does not modify the original dataset unless you save the result.

```
directmktg %>%
  select(userid, age, gender) %>%
  head()
```

```
    userid age gender
1 15624510  19   Male
2 15810944  35   Male
3 15668575  26 Female
4 15603246  27 Female
5 15804002  19   Male
6 15728773  27   Male
```

“Negative” selection can also be done by using the `-` (minus sign) before a variable name or a vector of variable names.

```
directmktg %>%
  select(-gender) %>%
  head()
```

```
    userid age buy salary age10 buy_binary salary_cat
1 15624510  19  No     19   1.9          0        Low
2 15810944  35  No     20   3.5          0        Low
3 15668575  26  No     43   2.6          0        Low
4 15603246  27  No     57   2.7          0        Med
5 15804002  19  No     76   1.9          0        Med
6 15728773  27  No     58   2.7          0        Med
```

```
directmktg %>%
  select(-c(age,gender)) %>%
  head()
```

```
    userid buy salary age10 buy_binary salary_cat
1 15624510  No     19   1.9          0        Low
2 15810944  No     20   3.5          0        Low
3 15668575  No     43   2.6          0        Low
4 15603246  No     57   2.7          0        Med
5 15804002  No     76   1.9          0        Med
6 15728773  No     58   2.7          0        Med
```

5.3 Filtering observations

Use `filter()` to keep rows that meet certain conditions. (Note: the `nrow(object_name)` from base R provides the number of rows in the data frame).

```
nrow(directmktg)
```

```
[1] 400
```

```
directmktg %>%
  filter(age >= 35) %>%
  nrow()
```

```
[1] 254
```

Multiple conditions can be combined:

```
directmktg %>%
  filter(age >= 35,
         gender == "Male") %>%
  nrow()
```

```
[1] 124
```

5.4 Creating new variables

Use `mutate()` to create or transform variables. New variables are added to the data frame.

```
directmktg %>%
  select(userid, age) %>%
  mutate(age10 = age / 10) %>%
  head()
```

```
    userid age age10
1 15624510  19   1.9
2 15810944  35   3.5
3 15668575  26   2.6
4 15603246  27   2.7
5 15804002  19   1.9
6 15728773  27   2.7
```

5.5 Summaries with `summarise()`

The `summarise()` function is used to compute **summary statistics** from a data frame. It can be used **with or without grouping**.

When `summarise()` is used **without** `group_by()`, it computes summaries over the entire dataset.

```r
directmktg %>%
  summarise(
    n = n(),
    mean_age = mean(age),
    buy_rate = mean(buy == "Yes")
  )
```

```
    n mean_age buy_rate
1 400   37.655   0.3575
```

Here's what this code is doing:

- `n()` counts the total number of observations in the dataset
- `mean(age)` computes the overall average age
- `mean(buy == "Yes")` computes the overall purchase rate

The result is a data frame with **one row**, where each column represents a summary statistic for the full dataset.

5.6 Grouped summaries with `group_by()` and `summarise()`

Often, you want to compute summaries **separately for different groups**, such as customer segments or demographic categories.

In `dplyr`, this is done by combining `group_by()` with `summarize()`.

```
directmktg %>%
  group_by(gender)
```

```
# A tibble: 400 × 8
# Groups:   gender [2]
     userid   age buy   gender salary age10 buy_binary salary_cat
      <dbl> <dbl> <fct> <fct>   <dbl> <dbl>      <dbl> <chr>
 1 15624510    19 No    Male       19   1.9          0 Low
 2 15810944    35 No    Male       20   3.5          0 Low
 3 15668575    26 No    Female     43   2.6          0 Low
 4 15603246    27 No    Female     57   2.7          0 Med
 5 15804002    19 No    Male       76   1.9          0 Med
 6 15728773    27 No    Male       58   2.7          0 Med
 7 15598044    27 No    Female     84   2.7          0 High
 8 15694829    32 Yes   Female    150   3.2          1 High
 9 15600575    25 No    Male       33   2.5          0 Low
10 15727311    35 No    Female     65   3.5          0 Med
# ℹ 390 more rows
```

The `group_by()` function does not change the data values. Instead, it tells R how the data should be **temporarily divided into groups** for the next operation.

At this point, no calculations have been performed. Once the data are grouped, `summarise()` computes statistics **within each group**.

```
directmktg %>%
  group_by(gender) %>%
  summarise(
    n = n(),
    mean_age = mean(age),
    buy_rate = mean(buy == "Yes")
  )
```

```
# A tibble: 2 × 4
  gender     n mean_age buy_rate
  <fct>  <int>    <dbl>    <dbl>
1 Male     196     36.9    0.337
2 Female   204     38.4    0.377
```

Step by step:

- `group_by(gender)` splits the data into separate groups based on gender
- `n()` counts observations within each group
- `mean(age)` computes the average age within each group
- `mean(buy == "Yes")` computes the purchase rate within each group

The result is a data frame with **one row per group** and **one column per summary statistic**.

After `summarise()` runs, the grouping structure is automatically dropped, so the result behaves like a regular data frame.

5.7 Connecting `summarize()` to base R

Conceptually, grouped summaries in `dplyr` perform the same task as a multi-step process in base R:

1. Split the data into groups

2. Compute summary statistics for each group
3. Combine the results into a table

For example, in base R you might compute group means using functions such as `aggregate()` or by manually subsetting the data.

The advantage of `group_by()` and `summarise()` is that these steps are expressed **explicitly and readably**, making your data transformations easier to follow, debug, and modify.

5.8 Combining transformations

As you've seen, one of the main advantages of `dplyr` is that multiple steps can be chained together.

```
directmktg_clean <- directmktg %>%
  filter(age >= 35) %>%
  mutate(age10 = age / 10,
         buy_binary = ifelse(buy == "Yes", 1, 0)) %>%
  select(userid, age10, gender, buy_binary)

head(directmktg_clean)
```

```
    userid age10 gender buy_binary
1 15810944   3.5   Male          0
2 15727311   3.5 Female          0
3 15733883   4.7   Male          1
4 15617482   4.5   Male          1
5 15704583   4.6   Male          1
6 15621083   4.8 Female          1
```

This approach keeps data preparation transparent and reproducible.

5.9 Key takeaway

Before any modeling or visualization, you should be able to: - load data reliably, - inspect variable types and values, - identify missing or problematic data, and - transform data into a usable analytical form.

These steps are essential for sound marketing analytics.

5.10 What’s next

In the next chapter, we will use cleaned and well-understood data to perform **descriptive analysis**, including frequency tables, crosstabs, measures of central tendency and dispersion, and correlation.