Chapter 5: Data Wrangling with dplyr Package

Overview: Once we loaded a dataset onto R, we may want to "prepare" the data before visualizing and/or analyze it. In Data Science, we called the process **data wrangling**. R has some built-in functions for data wrangling. The R package *dplyr*, is written specifically for data wrangling. It is part of the bigger packages *tidyverse*. When we install tidyverse, dplyr is also installed.

We can use *dplyr* to sort, filter, and manipulate data in a data frame.

Install the Packages: First, you must install the packages (your R should now be up-to-date and able to install these packages). Type the following into your R console (note: Don't type this in an R Markdown Chunk, you only need to run this line of code 1 time).

```
install.packages("dplyr")

# or

install.packages("tidyverse")  # if you want to dplyr and other packages
```

Load the Packages: You only need to install R packages one time. However, each time you start a new session (or a new markdown file), if you want to use any of these packages, you need to load them into your session. Type the following into your first chunk in your R markdown file:

```
library(dplyr)

# or

library(tidyverse)  # if you want to dplyr and other packages
```

Data: For this lesson, we will use the RealEstate data set. Download RealEstate.csv from Brightspace. Read that data set into your R session.

Using the dplyr Package: This package is used when you have a data set that you want to subset, organize, manipulate or create new columns from. To use different functions in this package, we use something called **piping** which we describe below.

Piping: Piping is a method that takes something (could be a variable or the output of a function) and feeds it into something else (usually a function). The general structure for piping is:

Example: Consider the following code:

```
x <- 9

x %>% sqrt()
```

We can also pipe multiple times. The same structure still follows where the item on the left gets piped into the next item to its right and so on:

Example: Consider the following code:

```
x <- c(9, 25)

x %>% sqrt() %>% sum()
```

Verbs: The functions in the *dplyr* package are called **verbs**. You can think of these as the actions that you wish to apply to the data. Combining this with a pipe, the typical structure of a command is:

We now go over some of the verbs that are part of the *dplyr* package:

1. rename() : We often need to refer to the column names of the data frame. We can rename the columns using the rename() verb.

   Example: Suppose we wanted to shorten the column names in the realEstate dataframe. We can do this with the following code:

```
colnames(realEstate)


realEstate %>% rename(date=col_names[2],age = col_names[3],
           distance = col_names[4],stores = col_names[5],
           latitude = col_names[6], longitude = col_names[7],
           price = col_names[8])

colnames(realEstate)
```

   Problem: We have renamed the columns of realEstate but we haven't saved this newly named data frame as anything so we can't refer back to it. To adjust this code so that we save it to a new data frame, we type:

```
realEstate2 = realEstate %>% rename(date=col_names[2],age = col_names[3],
              distance = col_names[4],stores = col_names[5],
              latitude = col_names[6], longitude = col_names[7],
              price = col_names[8])

colnames(realEstate2)
```

2. arrange() : This verb allows us to sort the values of a column in either ascending or descending order.

   Note: The default for this verb is to sort the values in ascending order (from smallest to biggest).

   For example:

```
realEstate2 %>% arrange(price)

realEstate2 %>% arrange(desc(price))
```

3. filter() : This verb allows you to subset the data frame based on some condition. The general form for using the filter verb is as follows:

   Example: Suppose we only wanted to consider houses that are older than 10 years. We could do this using filter:

```
realEstate2 %>% filter(age > 10)
```

   Practice Question: How many houses in the data frame are near 6 convenience stores?

   (A) 21 (B) 37 (C) 96 (D) 281

4. mutate(): This verb can change a variable (column) or add a new column.

   To change an existing column: The general form for the mutate verb is:

   Example: Consider the following code:

```
realEstate2 %>% mutate(age = age*365)
```

   To add a new column: The general form for the mutate verb is:

   Example: Consider the following code:

```
realEstate2 %>% mutate(dist_in_100m = distance/100)
```

New Data Set: For the next few examples, we will work with the *gapminder* data frame.

summarize(): This verb allows you to compute summarizing values for variables (such as mean or median or standard deviation). The general form for the summarize verb is:

Example: Consider the code:

```
gapminder %>% summarize(mean_pop = mean(pop))
```

Practice Question: Use the summarize command to determine the median life expectancy in the entire gapminder dataframe, calling it $med\_lifeExp$.

(A) 59.47444 (B) 60.7125 (C) 12.91711

Using multiple verbs (combining verbs): It is often the case that you want to use more than one verb. Since we use piping, the order in which you pipe the output of one verb into another matters.

Example: Suppose you want to determine the mean population for the year 1977. You first need to filter the data to only consider the year 1977, then you need to summarize the mean of the population. The following code accomplishes this:

Practice Question: Determine the median life expectancy for Canada (hint: combine the verbs filter and summarize).

(A) 74.985 (B) 74.90275 (C) 24499150

Practice Question: What are the two countries in Europe with the largest population in 2002?

(HInt: You will need to filter based on 2 variables and then arrange the population column in descending order)

(A) Germany and United Kingdom

(B) Turkey and United Kingdom

(C) Germany and Turkey

5. group_by() : This verb allows you to group categorical variables together (it is often used with other verbs or to help with plotting).

   Example: Consider the following code:

```
gapminder %>% group_by(country) %>% summarize(mean_pop = mean(pop))
```

   What is the output showing?

Extra Practice Problems from DataCamp:

The course 'Data Manipulation with dplyr' or 'Introduction to the Tidyverse' in DataCamp has very good short video explanations on how to use this package. They also use the gapminder dataset.