A Beginner Guide to Complete Data Wrangling with Dplyr In R Programming

There are different ways to perform data Wrangling in R. In this tutorial; we will use the dplyr package. It provides the most common data Wrangling tasks. That's why dplyr is called the grammar of data Wrangling. The main things related to data Wrangling using dplyr are filter, select, arrange, mutate, summarize, group, etc. Let's start one by one with the iris data set, a popular dataset for beginners.

First, we have to install Packages and then call it in R Environment.

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

We will deploy with the default iris dataset because it will better for any reader to import data and deploy it in their RStudio IDE. First of all, we have to import the dataset:

```
data(iris)
summary(iris)
```

Pipe operator: %>%: Before we go any further, let's introduce the pipe operator %>% because it will often use in our dplyr packages. dplyr imports this operator from another package (magrittr). This operator allows you to pipe the output from one function to the input of another function. For example, if you want to see the top 3 values of Sepal.Width you can run the following code.

```
iris %>% top_n(3, Sepal.Width)
```

The five major verbs in dplyr packages are described below:

select () select columns
filter ()filter rows
arrange () re-order or arrange rows
mutate() create new variables
summarise()summarize values
group_by()

filter (): Filtering datasets is one of the most common. For filtering, the filter () function comes in handy. For example, to see the Species contains the only setosa, we can run the following code:

```
filter <- filter (iris, Species == "setosa")
head(filter,3)
```

To see the Petal.Width less than or equal .2 we can run following code:

```
filter (iris, Petal.Width <= .2)
```

R offers the standard suite: >, >=, <, <= (less than or equal), != (not equal), and = = (equal).
select (): Often, you will work with large datasets with many columns, but suppose only a few columns are actually of interest to you or don't need all dataset columns for our analysis. You can select it magically using the select () function. In the following code, we select `Sepal.Length, Sepal.Width, Petal.Length` coloums from the dataset.

```
select <- select (iris, Sepal.Length, Sepal.Width, Petal.Length)
head(select)
```

Again. If you want to hide a particular column, the following code will be the best practice.

```
select <- select (iris, -Sepal.Length, -Sepal.Width)
head(select)
```

Besides, To select all columns that start or end with the character string "S," use the function starts_with()

```
select(iris, starts_with("S"))
select(iris, ends_with("S"))
```

You can use c() to concatenate vectors.

```
select (iris, -c (Petal.Width, Sepal.Width,Sepal.Length))
```

arrange (): Sometimes, we want your data ordered by a specific column value. We could sort the dataset according to the lowest sepal length to the highest length of the sepal. We could also sort the dataset by sorting the data frame for sepal length and then for sepal width:

```
arrange(iris, Sepal.Length)
arrange(iris, Sepal.Length, Sepal.Width)
```

If we want to sort the data frame in ascending order for sepal length but descending order for sepal width, Use desc() if you order a column in descending order:

```
arrange (iris,  desc(Sepal.Width))
```

To arrange rows by a particular column, such as the Sepal.Width, list the name of the column you want to arrange the rows by.

```
iris %>% arrange (Sepal.Width)
```

mutate (): If you want to make a new column/new variables from your existing columns of dataset you can handled and this is the job of mutate().

```
col1 <- mutate(iris, Greater.Half = Sepal.Width > 0.5 * Sepal.Length)
head(col1)
mutate(iris, New_col = Sepal.Length * 2)
```

summarise(): Summary statistics become much more powerful when combined with grouping. We could summarize different variables based on different summary statistics using the summarise() function. Here, we summarized the length and width of sepal and petal by calculating their average:

```
summarise(iris, meanSL=mean(Sepal.Length),
meanSW=mean(Sepal.Width),
meanPL=mean(Petal.Length),
meanPW=mean(Petal.Width))
```

Again the n() function in dplyr returns the number of rows.

```
iris %>% summarise(C=n())
```

group_by(): Sometimes, you want certain operations performed on groups in your dataset. Most data operations are done on groups defined by variables group_by(). You can group by multiple columns too:

```
iris %>%
group_by(Species)
```

Some other important function:
count (): Count the occurrence of each unique value in department

```
iris %>% count(Sepal.Width>=3.1)
```

slice(): Select certain rows based on row number - dplyr's **slice().** Here we select row from 140 to 150.

```
slice(iris , 140:150)
```

rename(): To rename the variables with new name rename() is good option for you.

```
rename(iris, SL=Sepal.Length , SW=Sepal.Width)
```

Conclusion: Today, you've learned how to analyze data with R's dplyr. It's one of the more friendly packages and simpler than its Python competitor – Pandas. We should be able to analyze and prepare any dataset after reading this article. You can do more advanced things and please search? dplyr in R console to see documentation of dplyr packages or search in google for advance dplyr guideline. Thank You.