Easy Guide (dplyr)

Sharon Chemweno

Introduction

In this vignette, we will; 1. Start with basics - what is a dplyr and its general form; 2. Then we will look at data wrangling by using select, mutate, arrange, filter, %>%, group_by and summarise

We will use 'USArrests' available in R. It is made up of arrests per one hundred thousand residents, for murder, assault and rape in each of the fifty states in the USA. The data is based on 1973 and the percentage of population living in the Urban areas is also included.

The goal of this documentation is help you to understand the uses of various functions in the Dplyr package.

What is dplyr?

Dplyr package was created by Hadley Wickham as an improvement from the existing package 'plyr'.It is designed to provide an extremely optimized set of routines for dealing with data frames. Dplyr is easy to use and highly useful in manipulating data frames. The package was designed to mitigate problems in R whose solutions is clear and easy to follow.

Some of the key functions provided in the package are: 1. mutate: Adds new variables/columns or transform existing variables. 2. arrange: Helps to reorder rows of a data frame 3. filter: Extracts rows from a data frame based on certain conditions 4. rename: Renames variables in a data frame 5. summarise/summarize: Generates summary of variables in a dataset 6. %>%: (known as the 'pipe' operator) It connects multiple verb actions together into a pipeline. 7. select: Returns a subset of columns from a data set

Installing the dplyr package:

```
install.packages("dplyr")
#>
#> The downloaded binary packages are in
#>  /var/folders/wk/yh3_khbx68l5m3vzj8mm6rxm0000gp/T//RtmpSJLwfz/downloaded_packages
```

Load the package

After installing, the package is loaded using the following code:

```
library(dplyr)
```

Load dataset:

```
 data("USArrests")
 summary(USArrests)
#>      Murder          Assault         UrbanPop          Rape      
#>  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30  
#>  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07  
#>  Median : 7.250   Median :159.0   Median :66.00   Median :20.10  
#>  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23  
#>  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18  
#>  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00  
```

You can see some basic characteristics of the dataset with the dim() and str() functions.

```
 dim(USArrests)
#> [1] 50  4
str(USArrests)
#> 'data.frame':    50 obs. of  4 variables:
#>  $ Murder  : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
#>  $ Assault : int  236 263 294 190 276 204 110 238 335 211 ...
#>  $ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
#>  $ Rape    : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...
names(USArrests)
#> [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

Using select()

The select() function can be used in several ways;

1. To select a column

```
x<-select(USArrests,c(Murder, Assault))
head(x)
#>            Murder Assault
#> Alabama      13.2     236
#> Alaska       10.0     263
#> Arizona       8.1     294
#> Arkansas      8.8     190
#> California    9.0     276
#> Colorado      7.9     204
```

2. To specify a range of variable names.

```
y<-select(USArrests, Assault: Rape)
head(y)
#>            Assault UrbanPop Rape
#> Alabama        236       58 21.2
#> Alaska         263       48 44.5
#> Arizona        294       80 31.0
#> Arkansas       190       50 19.5
#> California     276       91 40.6
#> Colorado       204       78 38.7
```

3. To omit variables by using the negative sign.

```
z<-select(USArrests,-c(UrbanPop,Rape))
head(z)
#>            Murder Assault
#> Alabama      13.2     236
#> Alaska       10.0     263
#> Arizona       8.1     294
#> Arkansas      8.8     190
#> California    9.0     276
#> Colorado      7.9     204
```

4. To specify variable names based on patterns.

```
USArrests_subset1 <- select(USArrests, starts_with('A'))
head(USArrests_subset1)
#>            Assault
#> Alabama        236
#> Alaska         263
#> Arizona        294
#> Arkansas       190
#> California     276
#> Colorado       204
```

Using mutate()

The mutate function can be used in various ways in a data frame. It can be used to create new variables that are derived from existing variables.

```
USArrests_data<- mutate(USArrests, Assault_derived= Assault- mean(Assault, na.rm = TRUE))
head(USArrests_data)
#>            Murder Assault UrbanPop Rape Assault_derived
#> Alabama      13.2     236       58 21.2           65.24
#> Alaska       10.0     263       48 44.5           92.24
#> Arizona       8.1     294       80 31.0          123.24
#> Arkansas      8.8     190       50 19.5           19.24
#> California    9.0     276       91 40.6          105.24
#> Colorado      7.9     204       78 38.7           33.24
```

Using filter()

The filter() function is used to extract subsets of rows from a data frame.

The function can be used in several ways:

1. To extract a row wth a certain required value

For example, using the data extract the rows of the USArrests data frame where the UrbanPop is greater than 50).

```
new_x<-filter(USArrests, UrbanPop > 50)
head(new_x)
#>             Murder Assault UrbanPop Rape
#> Alabama       13.2     236       58 21.2
#> Arizona        8.1     294       80 31.0
#> California     9.0     276       91 40.6
#> Colorado       7.9     204       78 38.7
#> Connecticut    3.3     110       77 11.1
#> Delaware       5.9     238       72 15.8
summary(new_x)
#>      Murder          Assault         UrbanPop          Rape
#>  Min.   : 2.100   Min.   : 46.0   Min.   :51.00   Min.   : 7.80
#>  1st Qu.: 4.300   1st Qu.:110.0   1st Qu.:62.00   1st Qu.:15.80
#>  Median : 7.200   Median :159.0   Median :70.00   Median :21.00
#>  Mean   : 7.673   Mean   :169.5   Mean   :70.29   Mean   :21.98
#>  3rd Qu.:11.100   3rd Qu.:238.0   3rd Qu.:80.00   3rd Qu.:26.90
#>  Max.   :17.400   Max.   :335.0   Max.   :91.00   Max.   :46.00
```

2. To obtain rows from certain conditions;

```
New_y<-filter(USArrests, UrbanPop > 50 & Murder >7)
summary(New_y$UrbanPop)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   52.00   65.00   74.00   72.71   80.00   91.00
summary(New_y$Murder)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>    7.20    8.50   11.10   10.95   12.70   17.40
head(New_y)
#>            Murder Assault UrbanPop Rape
#> Alabama      13.2     236       58 21.2
#> Arizona       8.1     294       80 31.0
#> California    9.0     276       91 40.6
#> Colorado      7.9     204       78 38.7
#> Florida      15.4     335       80 31.9
#> Georgia      17.4     211       60 25.8
summary(y$UrbanPop)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>   32.00   54.50   66.00   65.54   77.75   91.00
```

Using arrange()

The arrange() function is used to reorder rows of a data frame according to one of the variables. Reordering rows of a data frame (while preserving corresponding order of other columns) is normally a pain to do in R. The arrange() function simplifies the process quite a bit.

The following examples shows how the arrange() function can be used:

1. Example 1

It can be used to arrange a column. Please note, its arranged in ascending order if not specified.

```
X<-arrange(USArrests, Murder)
head(select(X, Assault, Murder), 10)
#>                Assault Murder
#> North Dakota        45    0.8
#> Maine               83    2.1
#> New Hampshire       57    2.1
#> Iowa                56    2.2
#> Vermont             48    2.2
#> Idaho              120    2.6
#> Wisconsin           53    2.6
#> Minnesota           72    2.7
#> Utah               120    3.2
#> Connecticut        110    3.3
tail(select(X, Assault, Murder), 10)
#>                 Assault Murder
#> Nevada              252   12.2
#> Texas               201   12.7
#> North Carolina      337   13.0
#> Alabama             236   13.2
#> Tennessee           188   13.2
#> South Carolina      279   14.4
#> Florida             335   15.4
#> Louisiana           249   15.4
#> Mississippi         259   16.1
#> Georgia             211   17.4
```

2. Example 2

In descending order;

```
Y<-arrange(USArrests, desc(Murder))
head(select(Y, Assault , Murder), 10)
#>                 Assault Murder
#> Georgia             211   17.4
#> Mississippi         259   16.1
#> Florida             335   15.4
#> Louisiana           249   15.4
#> South Carolina      279   14.4
#> Alabama             236   13.2
#> Tennessee           188   13.2
#> North Carolina      337   13.0
#> Texas               201   12.7
#> Nevada              252   12.2
tail(select(Y, Assault , Murder), 10)
#>                Assault Murder
#> Connecticut        110    3.3
#> Utah               120    3.2
#> Minnesota           72    2.7
#> Idaho              120    2.6
#> Wisconsin          53    2.6
#> Iowa               56    2.2
#> Vermont            48    2.2
#> Maine              83    2.1
#> New Hampshire      57    2.1
#> North Dakota       45    0.8
```

Using rename()

The rename() function is designed to make this process easier.

```
mydata <- rename(USArrests, "Assault Arrests"=Assault, "Urban population"=UrbanPop, "Rape Arrests"=Rape
head(mydata)
#>            Murder Arrests Assault Arrests Urban population Rape Arrests
#> Alabama              13.2             236               58         21.2
#> Alaska               10.0             263               48         44.5
#> Arizona               8.1             294               80         31.0
#> Arkansas              8.8             190               50         19.5
#> California            9.0             276               91         40.6
#> Colorado              7.9             204               78         38.7
```

Using group_by()

The group_by() function is used to generate summary statistics from the data frame within strata defined by a variable. The group_by() function first sets up how you want to group your data. The general operation here is a combination of splitting a data frame into separate pieces defined by a variable or group of variables (group_by()), and then applying a summary function across those subsets (summarize()).

1. Grouping with one column

```
Murder <- group_by(USArrests, Murder)
summarise(Murder, mean(Assault), mean(Rape))
#> # A tibble: 43 x 3
#>    Murder `mean(Assault)` `mean(Rape)`
#>     <dbl>           <dbl>        <dbl>
#>  1    0.8            45           7.3
#>  2    2.1            70           8.65
#>  3    2.2            52          11.2
#>  4    2.6            86.5        12.5
#>  5    2.7            72          14.9
#>  6    3.2           120          22.9
#>  7    3.3           110          11.1
#>  8    3.4           174           8.3
#>  9    3.8            86          12.8
#> 10    4             145          26.2
#> # ... with 33 more rows
```

2. Grouping with mutliple columns

```
groupby_Murder_Assault <- group_by(USArrests, Murder, Assault)
summarise(groupby_Murder_Assault, n = n())
#> # A tibble: 50 x 3
#> # Groups:   Murder [43]
#>    Murder Assault     n
#>     <dbl>   <int> <int>
#>  1    0.8      45     1
#>  2    2.1      57     1
#>  3    2.1      83     1
#>  4    2.2      48     1
#>  5    2.2      56     1
#>  6    2.6      53     1
#>  7    2.6     120     1
#>  8    2.7      72     1
#>  9    3.2     120     1
#> 10    3.3     110     1
#> # ... with 40 more rows
```

Using %>%

The pipeline operator (%>%)is used to string together multiple dplyr functions in a sequence of operations. It takes the output from one function and feed it to the first argument of the next function. Its optimized in such a way to allow you to string operations in a left-to-right fashion, i.e. first(x) %>% second %>% third

We can see the full use of the %>% operator using the following examples:

1. Example 1

We can pipe the USArrests data frame to the function that will select two columns (Murder and Assault) and then pipe the new data frame to the function head() which will return the head of the new data frame.

```
USArrests %>%
select(Murder, Assault) %>%
head
#>            Murder Assault
#> Alabama      13.2     236
#> Alaska       10.0     263
#> Arizona       8.1     294
#> Arkansas      8.8     190
#> California    9.0     276
#> Colorado      7.9     204
```

2. Example 2

To see how it works with other functions such as arrange, select, filter we will select three columns from USArrests data, arrange the rows by the Murder and then arrange the rows by Assault And filter the rows where Murder is greater equals 8 and Assault is greater than 200.

```
USArrests %>%
select(Murder, Assault, UrbanPop) %>%
arrange(Murder, Assault) %>%
filter(Murder >= 8 & Assault>200)
#>                 Murder Assault UrbanPop
#> Arizona            8.1     294       80
#> California         9.0     276       91
#> Alaska            10.0     263       48
#> Illinois          10.4     249       83
#> New York          11.1     254       86
#> Maryland          11.3     300       67
#> New Mexico        11.4     285       70
#> Michigan          12.1     255       74
#> Nevada            12.2     252       81
#> Texas             12.7     201       80
#> North Carolina    13.0     337       45
#> Alabama           13.2     236       58
#> South Carolina    14.4     279       48
#> Louisiana         15.4     249       66
#> Florida           15.4     335       80
#> Mississippi       16.1     259       44
#> Georgia           17.4     211       60
```

3. Example 3

To Create a new columns using mutate(), we can add new columns to the data frame. Create a new column called Murder_Assault which is multiplication of Murder and Assault

```
USArrests<-USArrests %>%
mutate(Murder_Assault = Murder*Assault)
head(USArrests)[,c("Murder","Assault","Murder_Assault")]
#>            Murder Assault Murder_Assault
#> Alabama      13.2     236         3115.2
#> Alaska       10.0     263         2630.0
#> Arizona       8.1     294         2381.4
#> Arkansas      8.8     190         1672.0
#> California    9.0     276         2484.0
#> Colorado      7.9     204         1611.6
```

4. Example 4

To create summaries of the data frame using summarise() function for a given column in the data frame such as finding the mean. For example, to compute the average number of Murder, apply the mean() function to the column Murder and call the summary value Mean_Murder. There are many other summary statistics you could consider such sd(), min(), max(), median(), sum(), n() (returns the length of vector), first() (returns first value in vector), last() (returns last value in vector) and n_distinct() (number of distinct values in vector).

```
USArrests %>%
summarise(Mean_Murder=mean(Murder),Max_Assault=max(Assault), Min_Rape=min(Rape))
#>   Mean_Murder Max_Assault Min_Rape
#> 1       7.788         337      7.3
```

Conclusion

The dplyr package is extremely useful in data wrangling. Much of the data analysis lies in manipulating data to obtain what you are looking for. Its not a surprise that data scientists will find themselves using the dplyr package in their analysis.

References

McNeil, D. R. (1977) Interactive Data Analysis. New York: Wiley. Wickham, H. and Grolemund, G. (2017) R for Data Science. New York: O'reilly https://r4ds.had.co.nz/transform.html