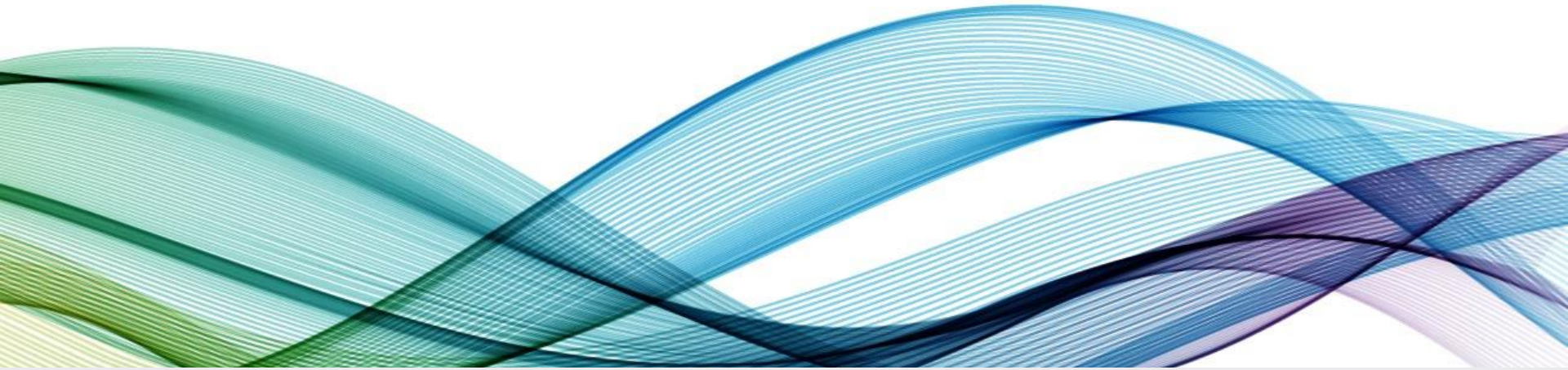
An Introduction to dplyr() package

Day 2



Agenda – Day 2

- In-built data sets in R
- Glimpse of In-built Iris dataset
- What is dplyr package?
- Six major functions in the package dplyr()
- Sub-setting : filter ()
- Selecting Columns : select ()
- Creating new columns : mutate()
- Ordering : arrange()
- Grouping & Summarizing : summarize() and group_by()
- Functional Pipelines : % > %

In-built data sets in R

- In this section we are going to discuss about the in-built datasets that R provides us with.

- To access the in-built datasets, we have to install a package called “dataset”. Once you have installed your datasets package, you need to call your package using the library of datasets
- Once you call your datasets package, you can easily take a look at all the in-built datasets that are available

- Of the in-built dataset available we are going to analyze a dataset called iris (using dplyr package). The way to call your iris dataset would be data (iris).

Glimpse of In-built Iris dataset

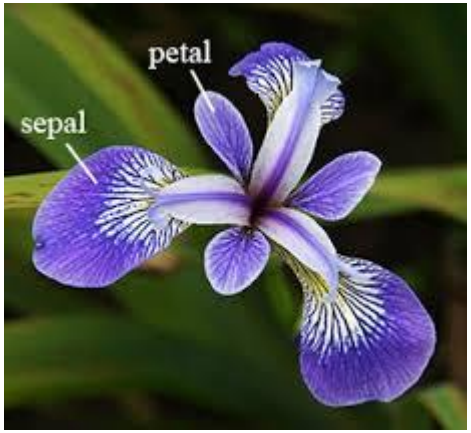
150 Observations

S.No	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
...					
50	5	3.3	1.4	0.2	setosa
...					
150	5.9	3	5.1	1.8	virginica

Numeric Variables

Categorical Variable

Attributes / Variables



Species :

Iris setosa

Iris versicolor

Iris virginica

What is dplyr package?

- dplyr is a very powerful data manipulation package and in recent days it has been gaining lot of popularity

  Please Note: dplyr works only with data frames

- In consecutive slides we are going to discuss six major functions in the package ‘dplyr’ that covers pretty much the whole spectrum of data manipulation tasks.

- We will also discuss how these functions can be combined together using pipeline to do complicated manipulation tasks using just a line of code.

Six major functions in the package dplyr()

- filter () : select columns 
- select () : filter rows 
- mutate () : To create a new column from the existing column 
- arrange () : To arrange data in ascending/descending order 
- group_by () : To group the data 
- summarize () : To summarize the data 

Note : In order to do group wise summary we have to use summarize () and group_by () functions together

Sub-setting : filter ()

- Let us discuss the first deplyr function filter ( ), which is used to subset the data
- Let us install and load the dplyr package

```
install.packages("dplyr")

library(dplyr)
```

- Suppose if we want to select all the rows where the Species is setosa we can do this by using filter ()
- This is how we have to use the filter ()

```
> data1=filter(iris,Species=="setosa")
```

- Notice that the first argument in the filter () is the dataset name which is iris and then we have the logical statement
- We have created a new object data1 by sub-setting the iris data

Output:

```
> data1
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
49          5.3         3.7          1.5         0.2  setosa
50          5.0         3.3          1.4         0.2  setosa
```

Selecting Columns : select ()

- Suppose if we want to see only the columns / variables Sepal.Length and Sepal.Width we can do this by using the select ()
- This is how we have to use the select ()

```
data2=select(iris,Sepal.Length,Sepal.Width)
```

- Notice again that the first argument of the select () is the dataset name which is iris followed by the columns / variables to be selected
- Also note that in the out put given below we can see only two variables that we have used inside the select statement

Output:

```
> data2
    Sepal.Length Sepal.Width
1            5.1         3.5
2            4.9         3.0
3            4.7         3.2
4            4.6         3.1
5            5.0         3.6
6            5.4         3.9
7            4.6         3.4

145          6.7         3.3
146          6.7         3.0
147          6.3         2.5
148          6.5         3.0
149          6.2         3.4
150          5.9         3.0
```

Please Note : By using "-" operator inside select statement we can drop columns

Syntax:

```
> data2=select(iris,-Petal.Length, -Petal.Width, -Species)
```

Creating new columns : mutate()

- Suppose if we want to create a new column Log.Sepal.Length from the existing dataset, we can do so by using the mutate()
- This is how we have to use the mutate ()

```
> data3=mutate(iris,Log.Sepal.Length=log(Sepal.Length))
```

- As you might have noticed the first argument is again the name of the dataset, Log.Sepal.Length is the name of the new column that we want to create and we are storing the log of Sepal.Length which is in the iris dataset
- Note that in the out put given below we can see a new column Log.Sepal.Length

Output:

```
> data3
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species Log.Sepal.Length
1            5.1         3.5          1.4         0.2     setosa         1.629241
2            4.9         3.0          1.4         0.2     setosa         1.589235

149          6.2         3.4          5.4         2.3  virginica         1.824549
150          5.9         3.0          5.1         1.8  virginica         1.774952
```

Ordering : arrange()

- If we want to arrange the data in ascending order based on the variable Sepal.Length, we can do so by using the function arrange()
- This is how we have to use the arrange ()

```
> data4=arrange(iris,Sepal.Length)
```

- Again notice that the first argument is the name of the dataset followed by the variable name on which basis the data has to be sorted
- Note that in the out put given below we can see that the column Sepal.Length has values in ascending order

Output:

```
> data4
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            4.3         3.0          1.1         0.1     setosa
2            4.4         2.9          1.4         0.2     setosa
3            4.4         3.0          1.3         0.2     setosa
4            4.4         3.2          1.3         0.2     setosa
5            4.5         2.3          1.3         0.3     setosa

148          7.7         2.8          6.7         2.0  virginica
149          7.7         3.0          6.1         2.3  virginica
150          7.9         3.8          6.4         2.0  virginica
```

Please Note : If we want to arrange the data in descending order then use the syntax given below

Syntax:

```
> data4=arrange(iris,desc(Sepal.Length))
```

Grouping & Summarizing : summarize() and group_by()

- Suppose if we want to find the mean and standard deviation of Sepal.Length based on Species.
- First we have to group the data based on Species and store the grouped data under Gr_Species and then we have to use the summarize() on the grouped data to compute the mean and the standard deviation of Sepal.Length

```
> Gr_Species=group_by(iris,Species)
> summarize(Gr_Species,mean(Sepal.Length),sd(Sepal.Length))
```

- Note that in the output give below, we have mean and the standard deviation of Sepal.Length for all the three Species setosa, versicolor and virginica

Output:

```
     Species `mean(Sepal.Length)` `sd(Sepal.Length)`
      <fctr>                <dbl>              <dbl>
1     setosa                5.006          0.3524897
2 versicolor                5.936          0.5161711
3  virginica                6.588          0.6358796
```

```
> data4=arrange(iris,desc(Sepal.Length))
```

Functional Pipelines : % > %

- Will now discuss about functional pipelines %>% and see how the pipeline operator simplifies the syntax and makes it simpler for us to do powerful data manipulation tasks in a single of code
- Suppose if we want to find the average Sepal.Length of all the Species whose Petal.Length is >= 1.4
- dplyr code:

```
> summarize(filter(iris,Petal.Length>=1.4),mean(Sepal.Length))
```

Outputs :

```
  mean(Sepal.Length)
1           5.920144
```

- Pipeline operator code :

```
> iris%>%filter(Petal.Length>=1.4)%>%summarize(mean(Sepal.Length))
```

```
  mean(Sepal.Length)
1           5.920144
```

The first argument to the function can now be written to the left of the function rather than being written inside the parenthesis

Iris is the first argument to the filter () and it is written to the left of the pipe operator. If you see the second pipe operator, the subsetted data is to the left of it. So, it becomes the input for the summarize ().So the Sepal.Length column here belongs to the data subsetted by the filter part of the code

Note : The whole code can be written as take the data set iris > filter it based on Petal.Length > take the subsetted data > summarize it based on the column Sepal.Length