Package dplyr

Data Manipulation and Summarization

Contents

Introduction

- dplyr is a powerful package to transform and summarize a data frame or data frame like object.
- The package contains a set of functions that are very handy and easy when performing exploratory data analysis and data manipulation.
- dplyr is part of core tidyverse.
- dplyr is developed by Hadley Wickham.
- dplyr is known as grammar of data management and functions in this package as verbs.

Data Snapshot

basic_salary data consist salary of each employee with it's Location & Grade.

Variables

Observations

First_Name	Last_Name	Grade	Location	ba	ms
Alan	Brown	GR1	DELHI	17990	16070

Columns	Description	Type	Measurement	Possible values
First_Name	First Name	character	-	-
Last_Name	Last Name	character	-	-
Grade	Grade	character	GR1, GR2	2
Location	Location	character	DELHI, MUMBAI	2
ba	Basic Allowance	numeric	Rs.	positive values
ms	Management Supplements	numeric	Rs.	positive values

Creating Subset - filter()

- `filter()` allows us to select a subset of rows in the data frame.
- Multiple conditions may be applied to the data frame to obtain the desired subset.

```
# Install and load package dplyr
# Import basic_salary data
```

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

data<-read.csv("basic_salary.csv",header=TRUE)
```

Creating Subset - filter()

```
# To create a subset of GR1 employees in MUMBAI

head(filter(data,Grade=="GR1",Location=="MUMBAI"))

# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Rajesh	Kolte	GR1	MUMBAI	19250	14960
2	Neha	Rao	GR1	MUMBAI	19235	15200
3	Aaron	Jones	GR1	MUMBAI	23280	13490

filter() creates a subset of the dataframe with rows that meet the required specifications

Creating Subset - select()

- We often work with large data frames with many variables. Sometimes it may happen that only a few variables turn out to be useful.
- `select ()` helps us get subset of only the useful variables.

```
# To create a subset having variables First_Name,Grade,Location
s1<-select(data,First_Name,Grade,Location)
head(s1)

# Output
```

	First_Name	Grade	Location
1	Alan	GR1	DELHI
2	Agatha	GR2	MUMBAI
3	Rajesh	GR1	MUMBAI
4	Ameet	GR2	DELHI
5	Neha	GR1	MUMBAI
6	Sagar	GR2	MUMBAI

Only the variables in the list will be selected

Creating Subset - select()

```
# To create a subset of some columns together
s2<-select(data,First_Name:Grade)
head(s2)
# Output
```

	First_Name	Last_Name	Grade
1	Alan	Brown	GR1
2	Agatha	Williams	GR2
3	Rajesh	Kolte	GR1
4	Ameet	Mishra	GR2
5	Neha	Rao	GR1
6	Sagar	Chavan	GR2

To select all columns between two columns **semicolon (:)** can be used. This command will select all columns between **First_Name** and **Grade** (Both inclusive)

Creating Subset - select()

```
# To create a subset with not having some specified columns
s3<-select(data,-(Grade:ba))
head(s3)

# Output
```

```
  First_Name Last_Name    ms
1       Alan     Brown 16070
2     Agatha  Williams  6630
3     Rajesh     Kolte 14960
4      Ameet    Mishra  9300
5       Neha       Rao 15200
6      Sagar    Chavan  6700
```

All variables except those between **Grade** and **ba** (both inclusive) are selected.

Sorting Data - arrange()

- `arrange()` is used to reorder rows.
- assign the column names in the data frame that are to be sorted.

```
# Sort  ba in ascending order
a1<-arrange(data,ba)
head(a1)
```

Use **desc()** to order a column in descending order.

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Anup	Save	GR2	MUMBAI	11960	7880
2	Agatha	Williams	GR2	MUMBAI	12390	6630
3	Sagar	Chavan	GR2	MUMBAI	13390	6700
4	John	Patil	GR2	MUMBAI	13500	10760
5	Adela	Thomas	GR2	DELHI	13660	6840
6	Gaurav	Singh	GR2	DELHI	13760	13220

Sorting Data - arrange()

```
# Sort by multiple variables
a2<-arrange(data,Grade,Location,ba)
head(a2)

# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Alan	Brown	GR1	DELHI	17990	16070
2	Sneha	Joshi	GR1	DELHI	20660	NA
3	Neha	Rao	GR1	MUMBAI	19235	15200
4	Rajesh	Kolte	GR1	MUMBAI	19250	14960
5	Aaron	Jones	GR1	MUMBAI	23280	13490
6	Adela	Thomas	GR2	DELHI	13660	6840

Here data is sorted by Grade. Within a particular Grade, the data is sorted by Location. Within a particular Grade and Location, the data is sorted by ba. All sorting is done in ascending order.

Modifying Data - mutate()

- `mutate()` is used to add new columns to the data frame.
- These new columns are functions of the existing columns.

```
# To create a new column

m1<-mutate(data, tot=ba+ms)
head(m1)

# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms	tot
1	Alan	Brown	GR1	DELHI	17990	16070	34060
2	Agatha	Williams	GR2	MUMBAI	12390	6630	19020
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960	34210
4	Ameet	Mishra	GR2	DELHI	14780	9300	24080
5	Neha	Rao	GR1	MUMBAI	19235	15200	34435
6	Sagar	Chavan	GR2	MUMBAI	13390	6700	20090

If we want only the new column, use **transmute()**.

Summarizing Data - summarise()

`summarise()` function will give the summary statistics for a given column in the data frame.

```
# To get summary statistic of a variable
summarise(data,meanba=mean(ba,na.rm=TRUE),
          medianba=median(ba,na.rm=TRUE))
```

```
# Output
```

```
     meanba medianba
1 16154.58    14270
```

Some of the summary statistics could be **mean**, **median**, **max**, **min**, **sd**, **sum** .

Grouping Data - group_by()

- `group_by()` verb splits the data frame by assigned variables
- We can now apply functions to these individual groups

```
# To create group
# Get summary statistic

loc_wise<-group_by(data,Location)
summarise(loc_wise,count=n(),mean=mean(ba,na.rm=TRUE))

# Output
```

```
# A tibble: 2 x 3
  Location count   mean
  <fct>    <int>  <dbl>
1 DELHI        5 16170
2 MUMBAI       7 16144.
```

❑ Here, first use **group_by()** to get the distinct locations then use the function **summarise()** to get the count and meanba in the distinct locations

* Incase your summarise code gives an error, just execute the following :
dplyr::summarise(loc_wise,count = n(),mean=mean(ba,na.rm=TRUE))

Drawing Random Samples

- `sample_n()` and `sample_frac()` are used to draw random samples of rows from the rows of data frame
- `sample_n()` gives a fixed number of rows
- `sample_frac()` gives a fixed fraction

```
# To draw a random sample of 5 rows
data_5<-sample_n(data,5)
head(data_5)

# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Neha	Rao	GR1	MUMBAI	19235	15200
2	Alan	Brown	GR1	DELHI	17990	16070
3	Sagar	Chavan	GR2	MUMBAI	13390	6700
4	Sneha	Joshi	GR1	DELHI	20660	NA
5	John	Patil	GR2	MUMBAI	13500	10760

Drawing Random Samples

```
# To draw a of random sample of size 10% (0.10) of data

data_0.1<-sample_frac(data,0.1)
data_0.1

# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms
1	Aaron	Jones	GR1	MUMBAI	23280	13490

Merging Data – join()

- `join()` in dplyr will work on data frames, tibbles and tbl references.
- The different operations could be –
  - `left_join()`: All rows from first data are included
  - `right_join()`: All rows from second data are included
  - `inner_join()`: Included rows which are matched on common variable
  - `full_join()`: All rows from two data are included with appropriate NA's

We have already studied these functions in detail in DM 04 - Merging, Appending _ Aggregating Data

Pipe Operator

- dplyr imports Pipe operator (%>%) from another package (magrittr)
- It's performance is similar to that of nested function.
- This operator allows the output of one function to be taken as input for another function.
- The chain of functions are read from left to right.

```
# Syntax for pipe operator to get first 6 rows of selected variables
```

```
data %>%
  select(First_Name,Grade,Function) %>%
  head
```

```
# Output
```

	First_Name	Grade	Location
1	Alan	GR1	DELHI
2	Agatha	GR2	MUMBAI
3	Rajesh	GR1	MUMBAI
4	Ameet	GR2	DELHI
5	Neha	GR1	MUMBAI
6	Sagar	GR2	MUMBAI

Pipe Operator

```
# Syntax for pipe operator to get data of selected variables belonging
to MUMBAI
```

```
data %>%
  select(First_Name,Grade,Location) %>%
  filter(Location=="MUMBAI")
```

```
# Output
```

	First_Name	Grade	Location
1	Agatha	GR2	MUMBAI
2	Rajesh	GR1	MUMBAI
3	Neha	GR1	MUMBAI
4	Sagar	GR2	MUMBAI
5	Aaron	GR1	MUMBAI
6	John	GR2	MUMBAI
7	Anup	GR2	MUMBAI

Some More Functions-n_distinct()

- `n_distinct()` is used to count the number of unique values in a set of vectors.
- It is much faster than using `length(unique()).`

```
# To get the unique length
n_distinct(data)
```

```
[1] 41
```

```
n_distinct(data$Grade,na.rm=TRUE)
```

```
[1] 2
```

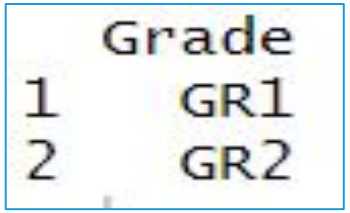
If we do not want to include **NA**, we use **na.rm=TRUE**

Some More Functions-distinct()

- `distinct()` retains only unique/distinct rows from an input tbl.
- This is similar to `unique.data.frame()`, but considerably faster.

```
# To get the unique categories in Grade
distinct(data,Grade)
```

```
# Output
```



	Grade
1	GR1
2	GR2

Some More Functions-dense_rank()

- Sometimes we may want not only to rank the dataset based on a variable but also display the rank along with the dataset.
- `dense_rank()` gives the ranking with no gaps between ranks.

```
# To sort on basis of a variable and to display rank alongwith
```

```
data %>%
  mutate(Rank=dense_rank(desc(ba)))%>%
  arrange(Rank) %>%
  head()
```

```
# Output
```

	First_Name	Last_Name	Grade	Location	ba	ms	Rank
1	Aaron	Jones	GR1	MUMBAI	23280	13490	1
2	Sneha	Joshi	GR1	DELHI	20660	NA	2
3	Rajesh	Kolte	GR1	MUMBAI	19250	14960	3
4	Neha	Rao	GR1	MUMBAI	19235	15200	4
5	Alan	Brown	GR1	DELHI	17990	16070	5
6	Ameet	Mishra	GR2	DELHI	14780	9300	6

Other than **`dense_rank()`**, the following functions also provide great help : **`row_number()`, `min_rank()`, `ntile()`, `percent_rank()`, `cume_dist()`**

Some More Functions-rowwise()

- Standard data frame operations in R are done column-wise rather than row-wise.
- `rowwise()` function is used when we want to perform row-wise operations (For instance, calculating row mean).

```
# To get the row means
data %>%
  rowwise()%>%
  mutate(meanval=mean(c(ba,ms))) %>%
  head(3)
```

```
# Output
```

```
# A tibble: 3 x 7
  First_Name Last_Name Grade Location    ba    ms meanval
  <fct>      <fct>     <fct> <fct>    <int> <int>   <dbl>
1 Alan       Brown     GR1   DELHI    17990 16070   17030
2 Agatha     Williams  GR2   MUMBAI   12390  6630    9510
3 Rajesh     Kolte     GR1   MUMBAI   19250 14960   17105
```

Some More Functions-select_if()

- `select_if()` drops variables that are not in the selection list
- This function is a variant of `select().`

```
# To select specific variables

data %>%
  select_if(is.numeric) %>%
  head()

# Output
```

	ba	ms
1	17990	16070
2	12390	6630
3	19250	14960
4	14780	9300
5	19235	15200
6	13390	6700

Some More Functions-rename_if()

- `rename_if()` changes the column names of only the variables in the selected list.
- This function is a variant of `rename()`.

```
# To rename the specified variables

data %>%
  rename_if(is.numeric,toupper) %>%
  head(3)
```

```
# Output
  First_Name Last_Name Grade Location    BA    MS
1       Alan     Brown   GR1    DELHI 17990 16070
2     Agatha  Williams   GR2   MUMBAI 12390  6630
3     Rajesh     Kolte   GR1   MUMBAI 19250 14960
```

Columns which are of type numeric, their names are changed to upper case

na.if() is also a useful function that can convert particular values from the data to NAs. na_if() is a translation of the SQL command NULL_IF. For instance, sometimes missing values are assigned values 999 or 0 and it may be required to recode them as NA.

Quick Recap

In this session, we learnt about package dplyr which is known as the Grammar of Data Management. This package contains a set of functions that are very handy and easy when performing exploratory data analysis and data manipulation

Imp verbs in dplyr package	• filter(), select() is used for creating subsets. • arrange() for sorting data • mutate() for modifying data • summarise() for summarising data. • group_by() splits the data frame by assigned variables
Drawing Random Sample	• sample_n(), sample_fact() are functions used for drawing random numbers
Merging Data	• left_join(), right_join(), inner_join(), full_join()
Pipe Operator	• dplyr imports `(%>%)` this operator from another package (magrittr). Its performance is comparable to nested function
Miscellaneous Functions	• n_distinct(), distinct(), tally(), dense_rank(), row_wise(), select_if(), rename_if() are some more functions which are easy to understand and use for data manipulation.