Package sqldf

Importing, Sorting, Subsetting, Modifying, Aggregating and Merging Data

Contents

1. Introduction
2. Importing Data
3. Sorting Data
4. Renaming Data
5. Subsetting Data
6. Aggregating Data
7. Merging Data

sqldf

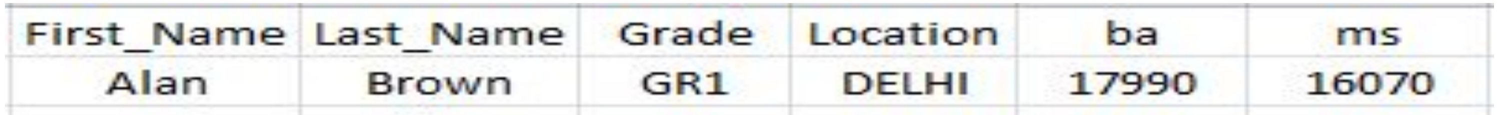
- For those who are learning R and may be well-versed in SQL, **package sqldf is very useful because it enables us to use SQL commands in R.**
- One who has basic SQL skills can **manipulate data frames and data tables in R using their SQL skills.**
- **SQL stands for Structured Query Language,** with data stored as tables in a database. There are number of database types, which are reasonably similar and sqldf uses SQLite as default.
- In this tutorial, we will see how to import, sort, modify, subset, aggregate and merge data in R using SQL.
- Let’s install the package and load the package to run SQL queries on R Data frames and data tables.

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

Data Snapshot

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

Variables



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

Observations

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

Importing data – read.csv.sql()

- `read.csv.sql()` function is used to import the datasets into R, filtering it with an SQL statement.

```
salary_data<-read.csv.sql("basic_salary.csv",sql="select * from
file",header=TRUE)
```

- **`file`** refers to the file given as first argument i.e 'basic_salary'.
- This function is very similar to other functions that allow importing datasets into R, with the sole exception that the second argument you pass is an SQL statement.
- Only the filtered portion is processed by R so that files larger than R can otherwise handle & can be accommodated.
- This function replaces the NA values with 0.

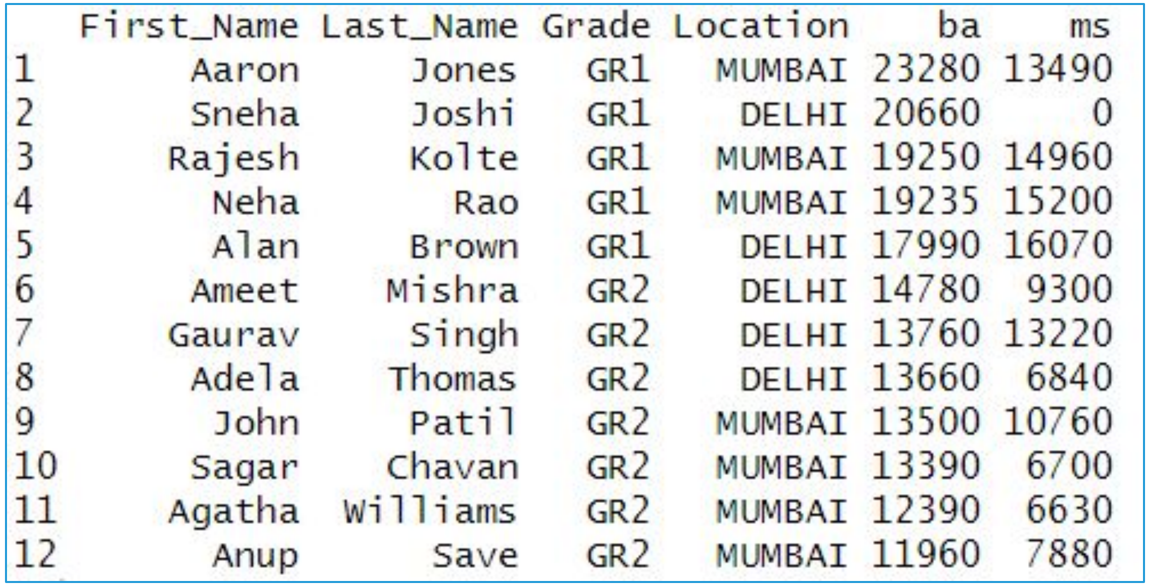
Sorting Data – order by

Sort salary_data by ‘ba’ in Descending order

```
# Sorting by one column
```

```
sortdata<-"select * from salary_data order by ba desc"
sorteddf<-sqldf(sortdata)
sorteddf
```

```
# Output
```



```
   First_Name Last_Name Grade Location    ba    ms
1       Aaron     Jones   GR1   MUMBAI 23280 13490
2       Sneha     Joshi   GR1    DELHI 20660     0
3      Rajesh     Kolte   GR1   MUMBAI 19250 14960
4        Neha       Rao   GR1   MUMBAI 19235 15200
5        Alan     Brown   GR1    DELHI 17990 16070
6       Ameet    Mishra   GR2    DELHI 14780  9300
7      Gaurav     Singh   GR2    DELHI 13760 13220
8       Adela    Thomas   GR2    DELHI 13660  6840
9        John     Patil   GR2   MUMBAI 13500 10760
10      Sagar    Chavan   GR2   MUMBAI 13390  6700
11     Agatha  Williams   GR2   MUMBAI 12390  6630
12       Anup      Save   GR2   MUMBAI 11960  7880
```

- **order by** clause is used to sort data base on one or more columns
- For sorting in descending order **desc** is to be specified after that particular column name. By default, it sorts data in ascending order.

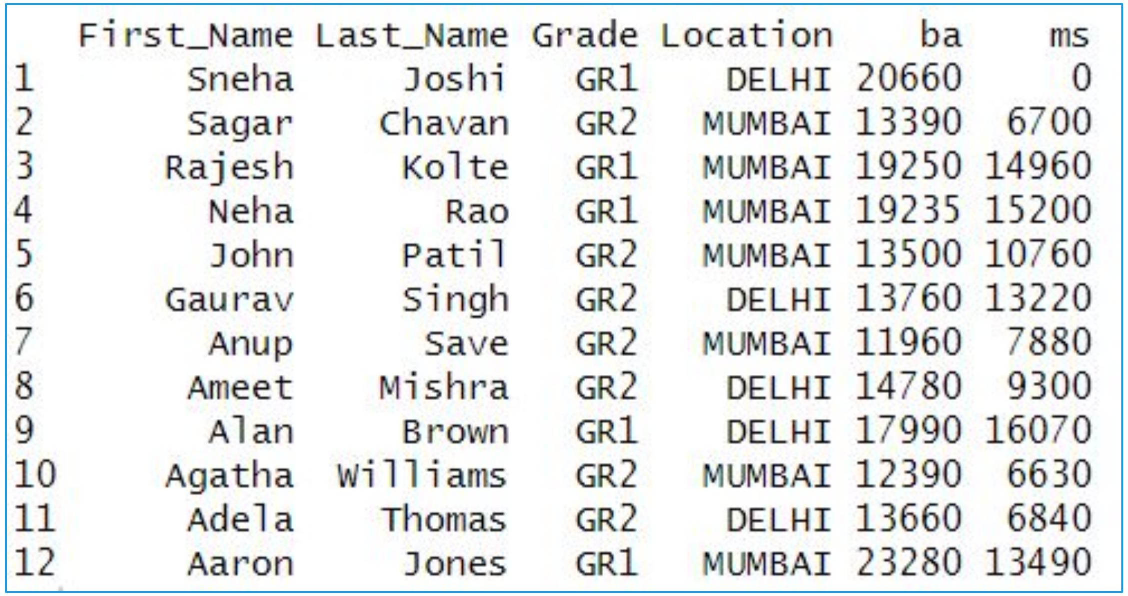
Sorting Data – order by

Sort salary_data by ‘First_Name’ in Descending order and ‘ba’ in Ascending order

```
# Sorting by multiple columns with different ordering levels
```

```
sortdata2<-"select * from salary_data order by First_Name desc,ba"
sorteddf2<-sqldf(sortdata2)
sorteddf2
```

```
# Output
```



```
   First_Name Last_Name Grade Location    ba    ms
1       Sneha     Joshi   GR1    DELHI 20660     0
2       Sagar    Chavan   GR2   MUMBAI 13390  6700
3      Rajesh     Kolte   GR1   MUMBAI 19250 14960
4        Neha       Rao   GR1   MUMBAI 19235 15200
5        John     Patil   GR2   MUMBAI 13500 10760
6      Gaurav     Singh   GR2    DELHI 13760 13220
7        Anup      Save   GR2   MUMBAI 11960  7880
8       Ameet    Mishra   GR2    DELHI 14780  9300
9        Alan     Brown   GR1    DELHI 17990 16070
10     Agatha  Williams   GR2   MUMBAI 12390  6630
11      Adela    Thomas   GR2    DELHI 13660  6840
12      Aaron     Jones   GR1   MUMBAI 23280 13490
```

Column names are specified by which data is sorted with **order by** clause.

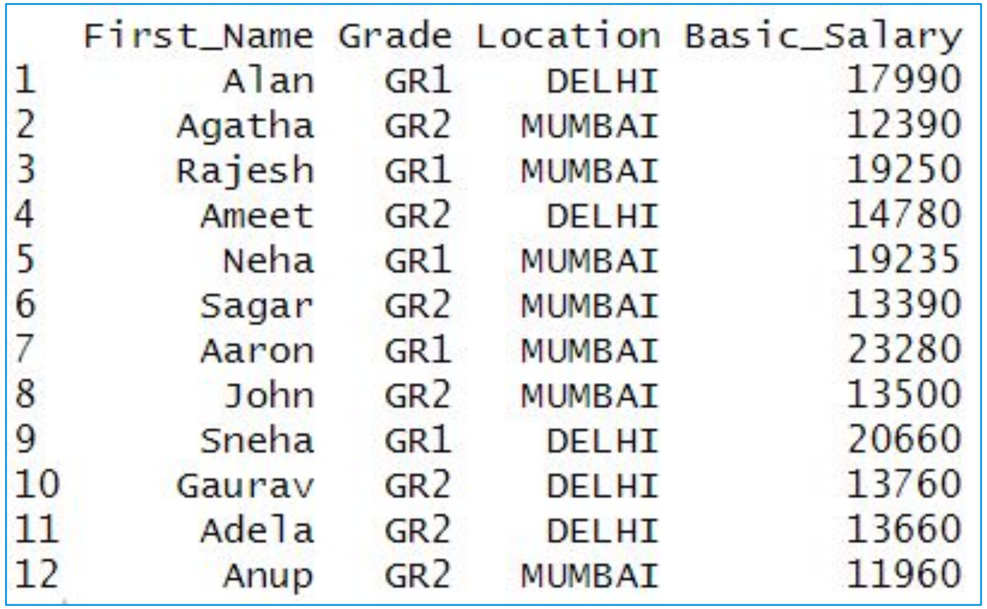
Renaming Columns - as

```
# Display columns First_Name,Grade,Location,ba of salary data and
# Rename column ‘ba’ as Basic_Salary
```

```
renamecols<-"select First_Name,Grade,Location,ba as Basic_Salary from
salary_data"
renamecolsdf<-sqldf(renamecols)
renamecolsdf
```

as clause is used with select statement to rename the columns in the output.

```
# Output
```



```
   First_Name Grade Location Basic_Salary
1        Alan   GR1    DELHI        17990
2      Agatha   GR2   MUMBAI        12390
3      Rajesh   GR1   MUMBAI        19250
4       Ameet   GR2    DELHI        14780
5        Neha   GR1   MUMBAI        19235
6       Sagar   GR2   MUMBAI        13390
7       Aaron   GR1   MUMBAI        23280
8        John   GR2   MUMBAI        13500
9       Sneha   GR1    DELHI        20660
10     Gaurav   GR2    DELHI        13760
11      Adela   GR2    DELHI        13660
12       Anup   GR2   MUMBAI        11960
```

Subsetting Data

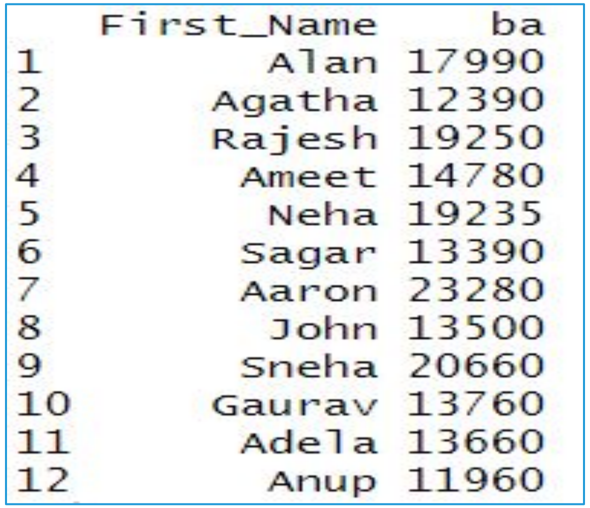
Display columns ‘First_Name’ and ‘ba’ of salary_data

```
# Column Subsetting
```

```
subcols<-"select First_Name,ba from salary_data"
subset_cols<-sqldf(subcols)
subset_cols
```

Column Subsetting is done by modifying the **select** statement by just giving the columns to be subsetted.

```
# Output
```

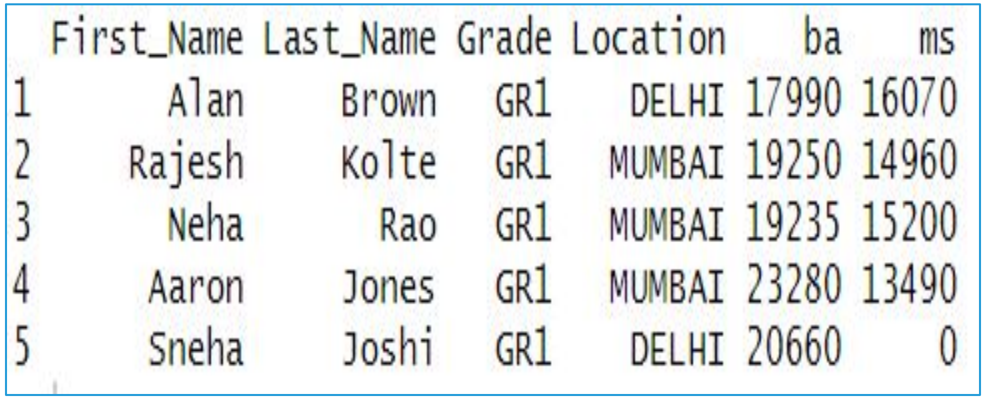


```
   First_Name    ba
1        Alan 17990
2      Agatha 12390
3      Rajesh 19250
4       Ameet 14780
5        Neha 19235
6       Sagar 13390
7       Aaron 23280
8        John 13500
9       Sneha 20660
10     Gaurav 13760
11      Adela 13660
12       Anup 11960
```

Subsetting Data

Show the records of employees having ‘ba’ more than 15000.

```
# Row Subsetting
subdata<-"select * from salary_data where ba>15000"
subset_ba<-sqldf(subdata)
subset_ba
# Output
```



```
  First_Name Last_Name Grade Location    ba    ms
1       Alan     Brown   GR1    DELHI 17990 16070
2     Rajesh     Kolte   GR1   MUMBAI 19250 14960
3       Neha       Rao   GR1   MUMBAI 19235 15200
4      Aaron     Jones   GR1   MUMBAI 23280 13490
5     Sneha      Joshi   GR1    DELHI 20660     0
```

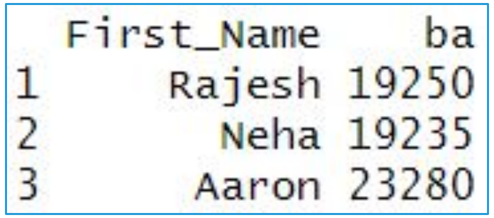
Row Subsetting is done by using **where** clause with a condition in SQL query

Subsetting Data

Display the ‘First_Name’ and ba of only those employees who are from MUMBAI ‘Location’ and having GR1 ‘Grade’.

```
sqldf("select First_Name,ba from salary_data where Location='MUMBAI'
and Grade='GR1'")
```

```
# Output
```



```
  First_Name    ba
1     Rajesh 19250
2       Neha 19235
3      Aaron 23280
```

By using **and** clause, we can give multiple conditions for subsetting data.

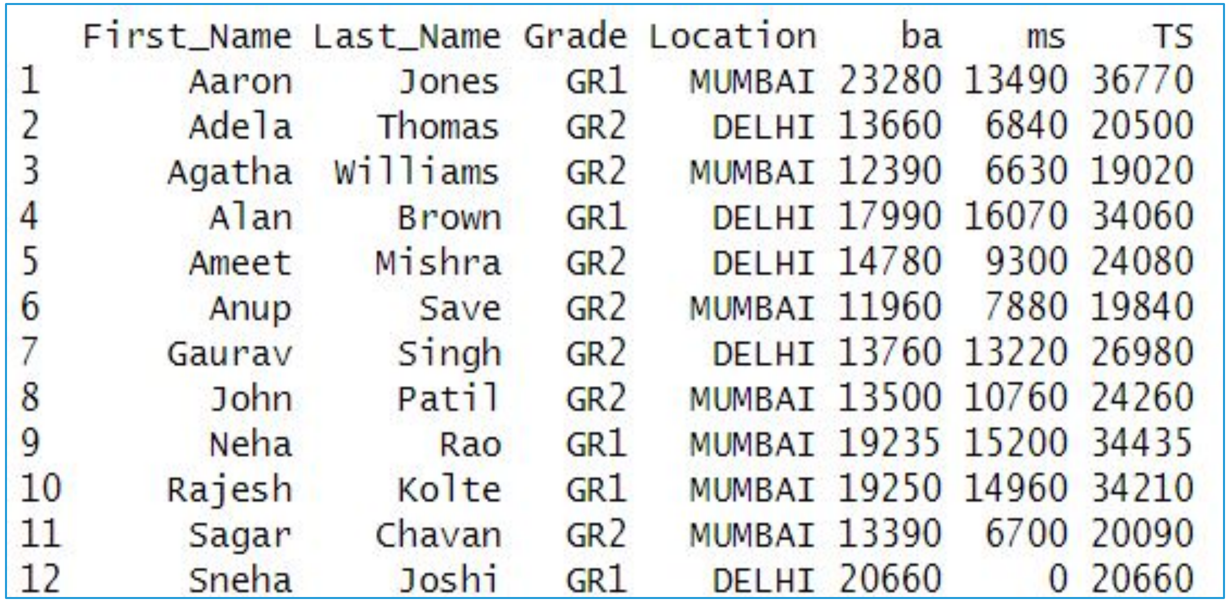
Subsetting Data

Calculate Total Salary of employees and display records of employees whose Total salary is greater than the median of Total salary of all employees

```
# Calculating Total Salary
```

```
new_saldata<-sqldf("select
First_Name,Last_name,Grade,Location,ba,ms,sum(ba+ms) as TS from
salary_data group by First_Name")
new_saldata
```

```
# Output
```



```
   First_Name Last_Name Grade Location    ba    ms    TS
1       Aaron     Jones   GR1   MUMBAI 23280 13490 36770
2       Adela    Thomas   GR2    DELHI 13660  6840 20500
3      Agatha  Williams   GR2   MUMBAI 12390  6630 19020
4        Alan     Brown   GR1    DELHI 17990 16070 34060
5       Ameet    Mishra   GR2    DELHI 14780  9300 24080
6        Anup      Save   GR2   MUMBAI 11960  7880 19840
7      Gaurav     Singh   GR2    DELHI 13760 13220 26980
8        John     Patil   GR2   MUMBAI 13500 10760 24260
9        Neha       Rao   GR1   MUMBAI 19235 15200 34435
10     Rajesh     Kolte   GR1   MUMBAI 19250 14960 34210
11      Sagar    Chavan   GR2   MUMBAI 13390  6700 20090
12     Sneha      Joshi   GR1    DELHI 20660     0 20660
```

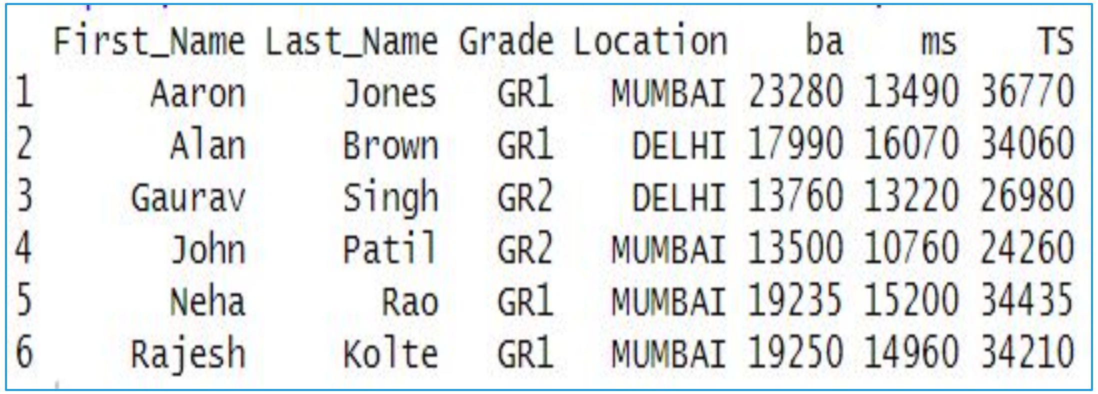
Here, we have used **sum()** function to calculate total salary and named that column as

Subsetting Data

```
# Record of employees whose TS > median TS of all the employees
```

```
sqldf("select * from new_saldata where TS>(select median(TS) from
new_saldata)")
```

```
# Output
```



```
  First_Name Last_Name Grade Location    ba    ms    TS
1      Aaron     Jones   GR1   MUMBAI 23280 13490 36770
2       Alan     Brown   GR1    DELHI 17990 16070 34060
3     Gaurav     Singh   GR2    DELHI 13760 13220 26980
4       John     Patil   GR2   MUMBAI 13500 10760 24260
5       Neha       Rao   GR1   MUMBAI 19235 15200 34435
6     Rajesh     Kolte   GR1   MUMBAI 19250 14960 34210
```

We can use multiple **select** statements inside the another. Here, we have used a **select** statement to calculate median of **TS** inside another **select** statement

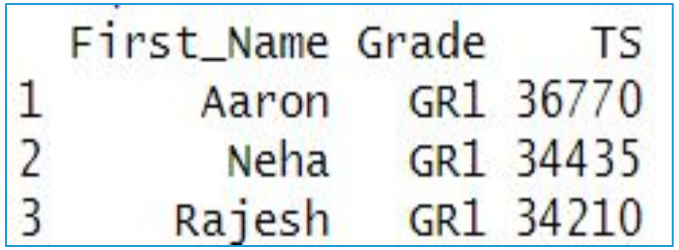
Subsetting Data

```
# Who are the Top 3 highest paid employees?
# Display their ‘First_Name’, ‘Grade’ and ‘TS’
```

```
sqldf("select First_Name,Grade,TS from new_saldata order by TS desc
limit 3")
```

limit statement is used to retrieve records and limit the number of records returned based on a limit value.

```
# Output
```



```
  First_Name Grade    TS
1      Aaron   GR1 36770
2       Neha   GR1 34435
3     Rajesh   GR1 34210
```

Interpretation :

- Above command sorts the data in the descending order of **TS** using **order by** clause and retrieves first 3 records from the data using **limit** statement.

Subsetting Data – More Examples

Few of the SQL queries for subsetting data:

```
# Subsetting rows using Logical operators (SQL and)
```

```
query1<-"select * from sal_data where Basic_Salary>14000 and
First_Name like 'r%'"
sqldf(query1)
```

This command will return the rows having First Name starting with ‘r’ and basic Salary greater than 14000. **SQL ‘or’** operator can also be used if we want either the first condition OR the second condition to be true.

```
# Subsetting rows by value range (SQL between)
```

```
query2<-"select * from sal_data where Basic_Salary between 16000 and
17000"
sqldf(query2)
```

This command will return all the records of employees whose Basic Salary is between 16000-17000.

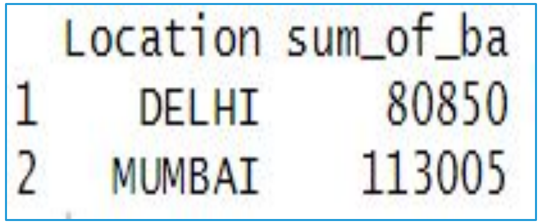
Aggregating Data – group by

```
# Calculate sum of variable ‘ba’ by variable ‘Location’
```

```
agg_sum<-"select Location,sum(ba) as sum_of_ba from salary_data group
by Location"
aggdf<-sqldf(agg_sum)
aggdf
```

- ❑ **as** clause is used to change the name of the column in the output.
- ❑ **group by** clause is used with select statement for aggregation in SQL.
- ❑ Aggregation functions like **sum(), count()** , **avg()** can be used with **select** statement.

```
# Output
```

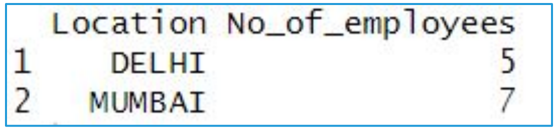


```
  Location sum_of_ba
1    DELHI     80850
2   MUMBAI    113005
```

```
# Display No. of Employees Location wise
```

```
agg_count<-"select Location,count(*) as No_of_employees from
salary_data group by Location"
countdf<-sqldf(agg_count)
countdf
```

```
# Output
```



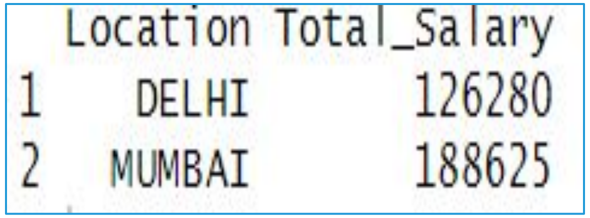
```
  Location No_of_employees
1    DELHI               5
2   MUMBAI               7
```

Aggregating Data – group by

```
# Aggregate TS(Total Salary) and show the percentage share of TS by
# ‘Location’
```

```
sum_ts<-sqldf("select Location,sum(TS) as Total_Salary from
new_saldata group by Location")
sum_ts
```

```
# Output
```

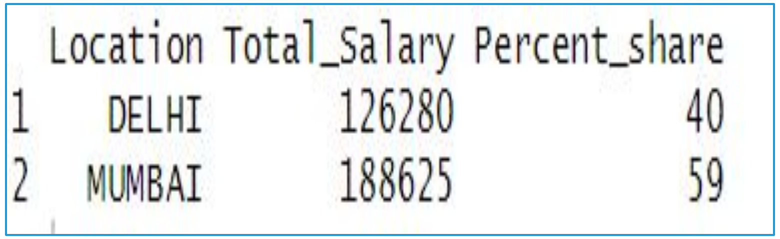


```
  Location Total_Salary
1    DELHI       126280
2   MUMBAI       188625
```

- This command calculates the sum of TS by Location and displays it with Location.
- Here we have used the data new_saldata which we created while subsetting based on TS.

```
sqldf("select Location,Total_Salary,(Total_Salary*100/(select
sum(Total_Salary)from sum_ts))as Percent_share from sum_ts")
```

```
# Output
```

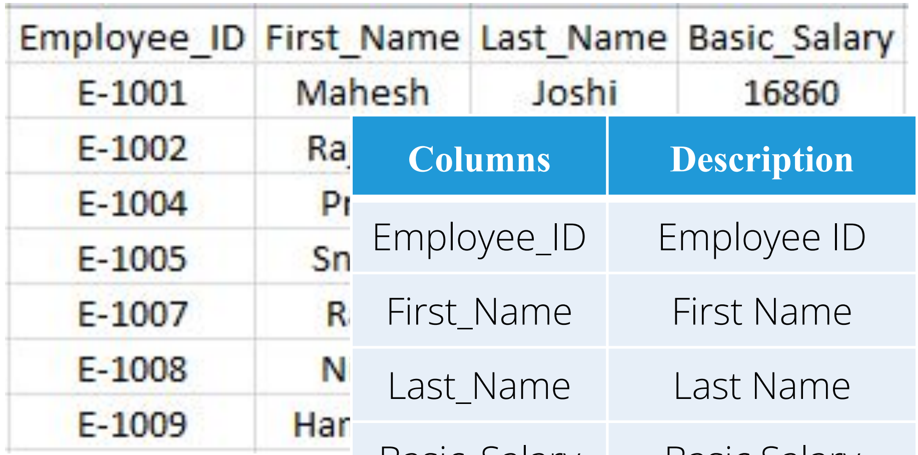


```
  Location Total_Salary Percent_share
1    DELHI       126280            40
2   MUMBAI       188625            59
```

This command displays **Location**, **Total Salary** & percentage share of **TS** by **Location**

Data Snapshot

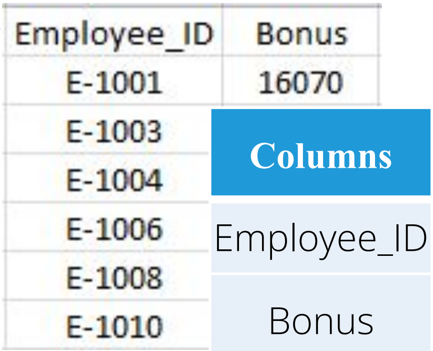
sal_data consist information about Employee's Basic Salary, their ID & full Name



Employee_ID	First_Name	Last_Name	Basic_Salary
E-1001	Mahesh	Joshi	16860
E-1002	Ra		
E-1004	Pr		
E-1005	Sn		
E-1007	R		
E-1008	N		
E-1009	Har		

Columns	Description	Type	Measurement	Possible values
Employee_ID	Employee ID	character	-	-
First_Name	First Name	character	-	-
Last_Name	Last Name	character	-	-
Basic_Salary	Basic Salary	numeric	Rs.	positive values

bonus_data has information of only Bonus given to Employees.



Employee_ID	Bonus
E-1001	16070
E-1003	
E-1004	
E-1006	
E-1008	
E-1010	

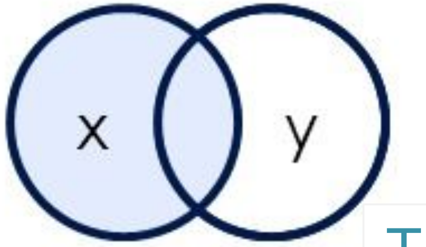
Columns	Description	Type	Measurement	Possible values
Employee_ID	Employee ID	character	-	-
Bonus	Bonus amount	Numeric	Rs.	Positive values

"Employee ID" is the common column in both datasets

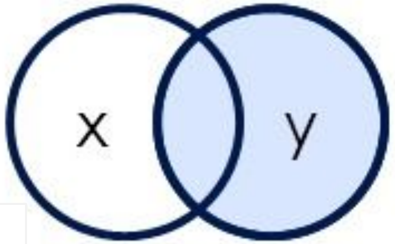
Merging Data-Types of Joins

Consider **sal_data =x** and **bonus_data=y**

Left Join

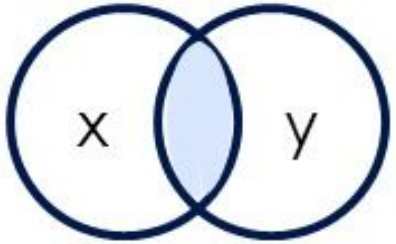


Right Join

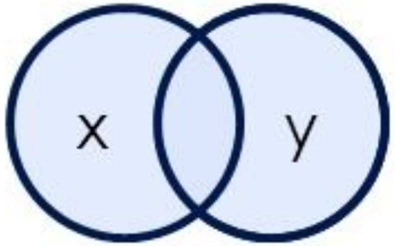


There are 4 types of Joins

Inner Join



Outer Join



Merging Data

- Performing joins is one of the most common operations in SQL.
- **Left Join** returns all rows from the left table, and any rows with matching keys from the right table whereas **Right Join** returns all rows from the right table, and any rows with matching keys from the left table.
- **Inner joins** return only rows with matching data for the common column, and **full outer joins** return all rows in all data sets, even if there are rows without matches.
- Currently, **sqldf** does not support right joins or full outer joins.
- Let's perform **left** and **inner join** on **sal_data** and **bonus_data.**
- Before performing joins first import two data sets **sal_data** and **bonus_data** and store them in objects as **sal_data** and **bonus_data.**

```
sal_data<-read.csv.sql("sal_data.csv",sql="select * from file",
header=TRUE)

bonus_data<-read.csv.sql("bonus_data.csv",sql="select * from file",
header=TRUE)
```

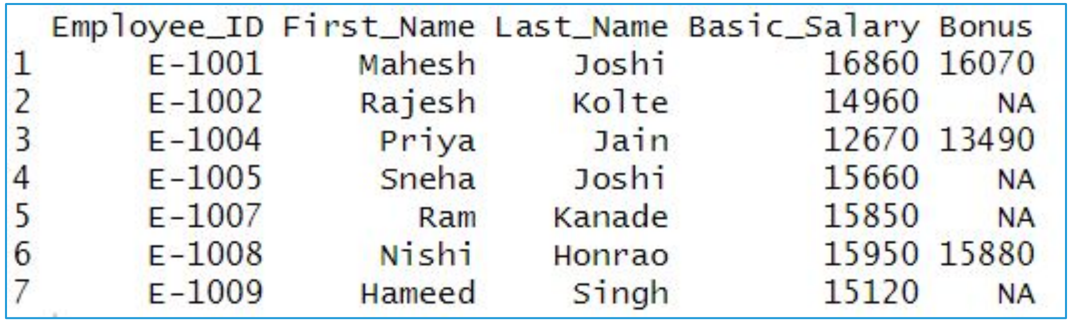
Merging Data – Left Join

```
# Left Join
```

```
leftjoin_string<-"select sal_data.*,bonus_data.Bonus from sal_data
left join bonus_data on sal_data.Employee_ID = bonus_data.Employee_ID"

sal_join_bonus<-sqldf(leftjoin_string)
sal_join_bonus
```

```
# Output
```



```
  Employee_ID First_Name Last_Name Basic_Salary Bonus
1      E-1001     Mahesh     Joshi        16860 16070
2      E-1002     Rajesh     Kolte        14960    NA
3      E-1004      Priya      Jain        12670 13490
4      E-1005      Sneha     Joshi        15660    NA
5      E-1007        Ram    Kanade        15850    NA
6      E-1008      Nishi    Honrao        15950 15880
7      E-1009     Hameed     Singh        15120    NA
```

A new data frame, **sal_join_bonus**, will be created using **sqldf().**
sqldf() at minimum, requires a character string with the SQL operation to be performed..

* `merge()` in base R performs the equivalent of inner and left joins, as well as right and full outer joins, which are unavailable in `sqldf`.

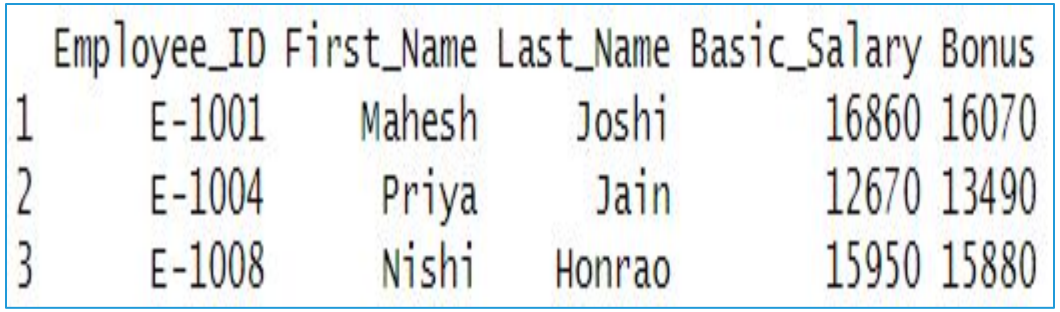
Merging Data – Inner Join

```
# Inner Join
```

```
innerjoin_string<-"select sal_data.*,bonus_data.Bonus from sal_data
inner join bonus_data on sal_data.Employee_ID=bonus_data.Employee_ID"

sal_join_bonus2<-sqldf(innerjoin_string)
sal_join_bonus2
```

```
# Output
```



```
  Employee_ID First_Name Last_Name Basic_Salary Bonus
1      E-1001     Mahesh     Joshi        16860 16070
2      E-1004      Priya      Jain        12670 13490
3      E-1008      Nishi    Honrao        15950 15880
```

Data management tasks using sqldf

Data management tasks using sqldf

- Importing data: Using **`read.csv.sql()`**
- Sorting data: Using **`order by`** clause
- Renaming columns: Using **`as`** clause
- Subsetting data: Using **`where`** clause
- Aggregating data: Using aggregation function on columns and **`group by`** clause
- Merging data: Left and inner join