Data Analytics with R

DATA AND
ARTIFICIAL INTELLIGENCE

Data Manipulation Using R

Business Scenario

- Anna was recently promoted to a junior data analyst in a credit card company. She is working on a customer retention strategy for which she needs to pull data from different sources.
- As a part of her job, she needs to collate, clean, and aggregate the data to understand customer value and prepare it for further study.

Approach: To efficiently perform her job, Anna needs to learn different aspects of data manipulation, including reading data tables from different sources, merging data tables, and summarizing them.

Learning Objectives

By the end of this lesson, you will be able to:

- List the steps of data wrangling
- Read and manipulate data using functions in R
- Summarize data
- Merge data tables to prepare data for analysis
- Use dplyr for data manipulation

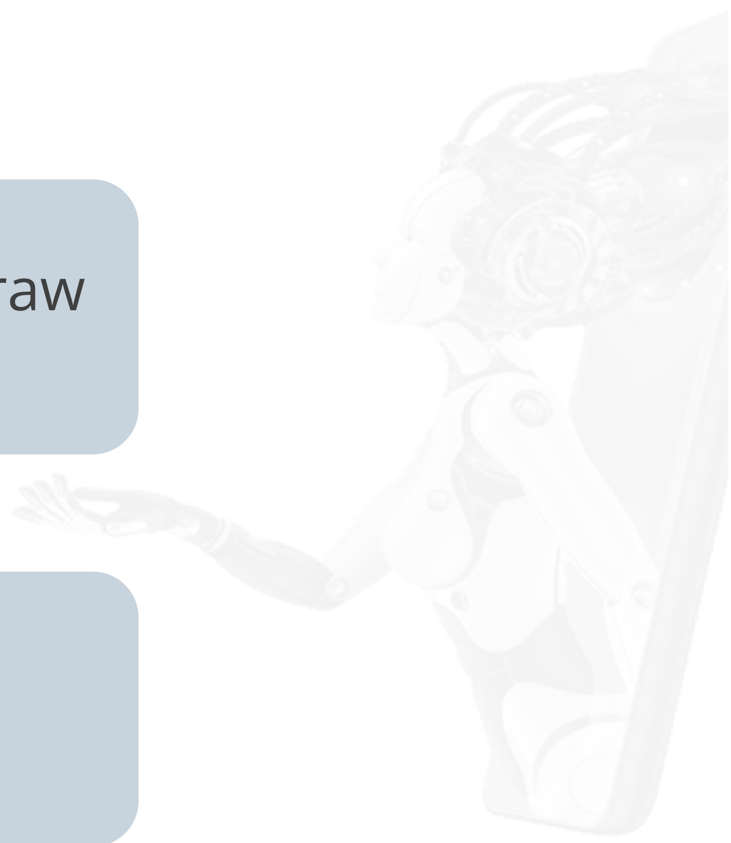
Data Wrangling

Data Wrangling

The outcome of an analysis is as good as the data being used.

Data wrangling is a combination of a variety of processes of transforming raw data to a more user-friendly format.

Data wrangling prepares the data for further exploration steps of data analysis.



Data Wrangling Steps

01	Discovery	Process of exploring the data in order to strategize how it can be used to solve the data analysis objective
02	Structuring	Process of transforming raw data into a more usable structured data per requirement of the analytical model to be used
03	Cleaning	Process of removing noise from the data that might adversely alter the outcome of the analysis
04	Enriching	Augment necessary data from other datasets which might add value to the present analysis
05	Validating	Process of verifying the data for consistency and quality
06	Publishing	Making the data available, that is, ready to be published

Reading Data in R

Reading Data Tables

R has functions to read data from a variety of formats such as text, Excel, SPSS stats data, and SAS datasets. For reading Excel data, read_excel() function is used from the readxl package.

Most of the functions to read data are available in the base package:

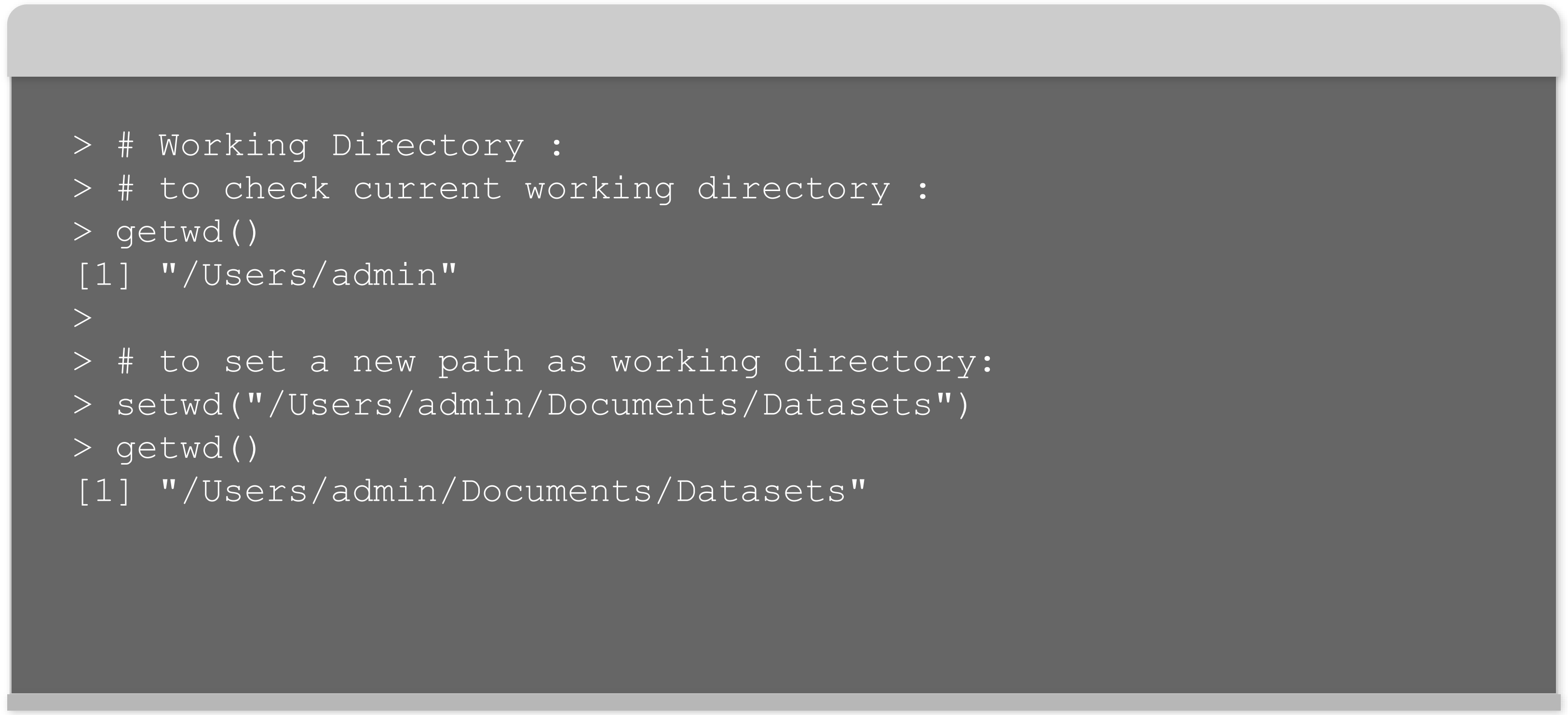
read.table()

read.csv()

read.delim()

Working Directory

The working directory is a location on a machine that sets the default location of any files to be read into R, or saved out of R.



```
> # Working Directory :
> # to check current working directory :
> getwd()
[1] "/Users/admin"
>
> # to set a new path as working directory:
> setwd("/Users/admin/Documents/Datasets")
> getwd()
[1] "/Users/admin/Documents/Datasets"
```

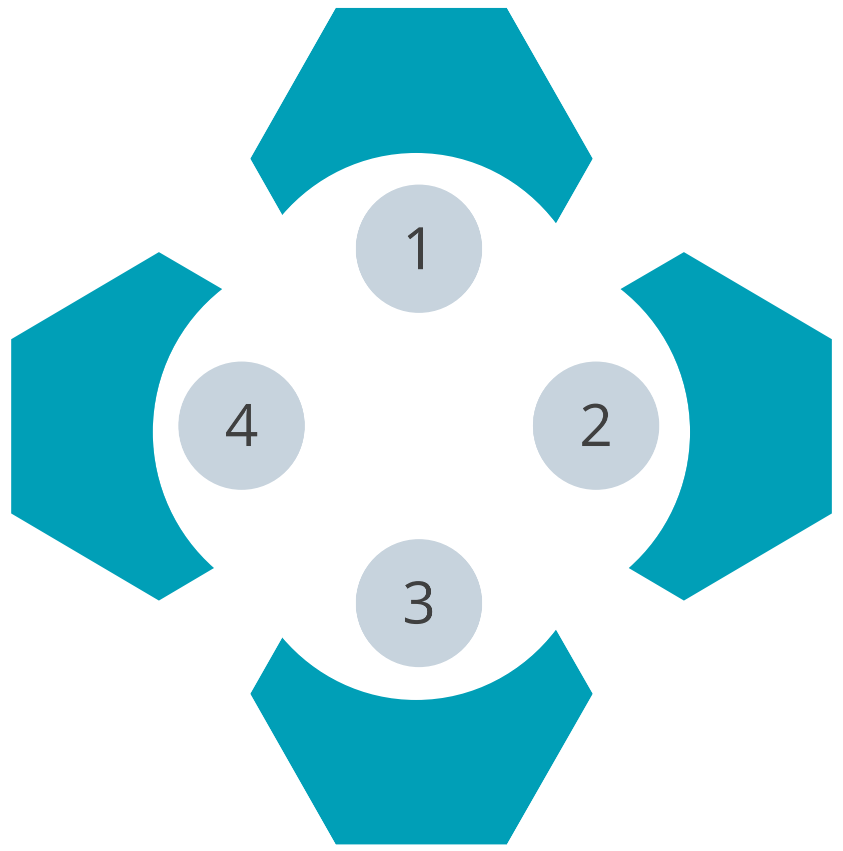
Reading a CSV File

The read.csv() function is used to read a “comma-separated value” format file in the R environment. It also reads the data table as a dataframe.

```
> # reading csv data :
> data <- read.csv('cars.csv’, header = TRUE, stringsAsFactors = TRUE)
> # printing first 5 rows and columns of the data
> print(data[1:5,1:5])
 Make Model Type Origin DriveTrain
1 Subaru Forester X Wagon All
2 Toyota Camry Solara SE V6 2dr Sedan Asia Front
3 Suzuki Aerio LX 4dr Sedan Asia Front
4 Dodge Dakota Club Cab Truck USA Rear
5 Mazda Mazda3 s 4dr Sedan Asia Front
```

Dataframe: Features

The column names should be non-empty.



Each column should contain same number of data items.

The row names should be unique.

The data stored in a dataframe can be of numeric, factor, or character type.

Reading an Excel File

An excel sheet can be imported in R environment by using read_excel function from the "readxl" package.

```
> # read excel data using readxl
> library(readxl)
> my_data <- read_excel('my_data.xlsx')
> my_data
# A tibble: 6 x 3
 id name score
 <chr> <chr> <dbl>
1 A101 Jack 56
2 A105 Stan 76
3 A108 Paul 44
4 A102 Dave 85
5 A106 Kim 64
6 A112 Leo 90
```

Note

read_excel() imports the data as a tibble.

Tibble

A tibble is a form of the dataframe that is a part of the tidy verse. It differs from dataframes on these aspects:

- A tibble does not change the input type.
- A tibble can have columns that are lists.
- A tibble can have non-standard variable names.
- A tibble does not create row names.

Dataframe: Basic Attributes

Function	Description
head(x, n = 6)	Returns top 6 rows of any dataframe, by default
tail(x, n = 6)	Returns bottom 6 rows of any dataframe, by default
str(x)	Returns the structural information of the data as number of rows, columns, names of columns, etc.
View(x)	Invokes a spreadsheet style data viewer for the object

Dataframe: Basic Attributes

Consider the following dataframe with 10 observations and five data columns.

```
> # consider the following dataframe with 10 rows
> name <- c('John','Allison','Claire','Allan',
'Debra','Jack','Reed', 'Arron', 'Paul', 'Stephanie')
> age <- c(32,25,27,28,22,35, 28, 31, 44, 29)
> sex <- c('M','F','F','M','F','M','F', 'M', 'M', 'F')
> height <- c(1.95,1.83, 1.78, 1.92, 1.87,1.75, 1.56, 1.67, 1.59,
1.77)
> weight <- c( 69, 73, 71, 77, 75, 78, 66, 72, 65, 70)
> df <- data.frame(name, age, sex, height, weight)
```

Snapshot of Data: head() & tail() Function

head(df)

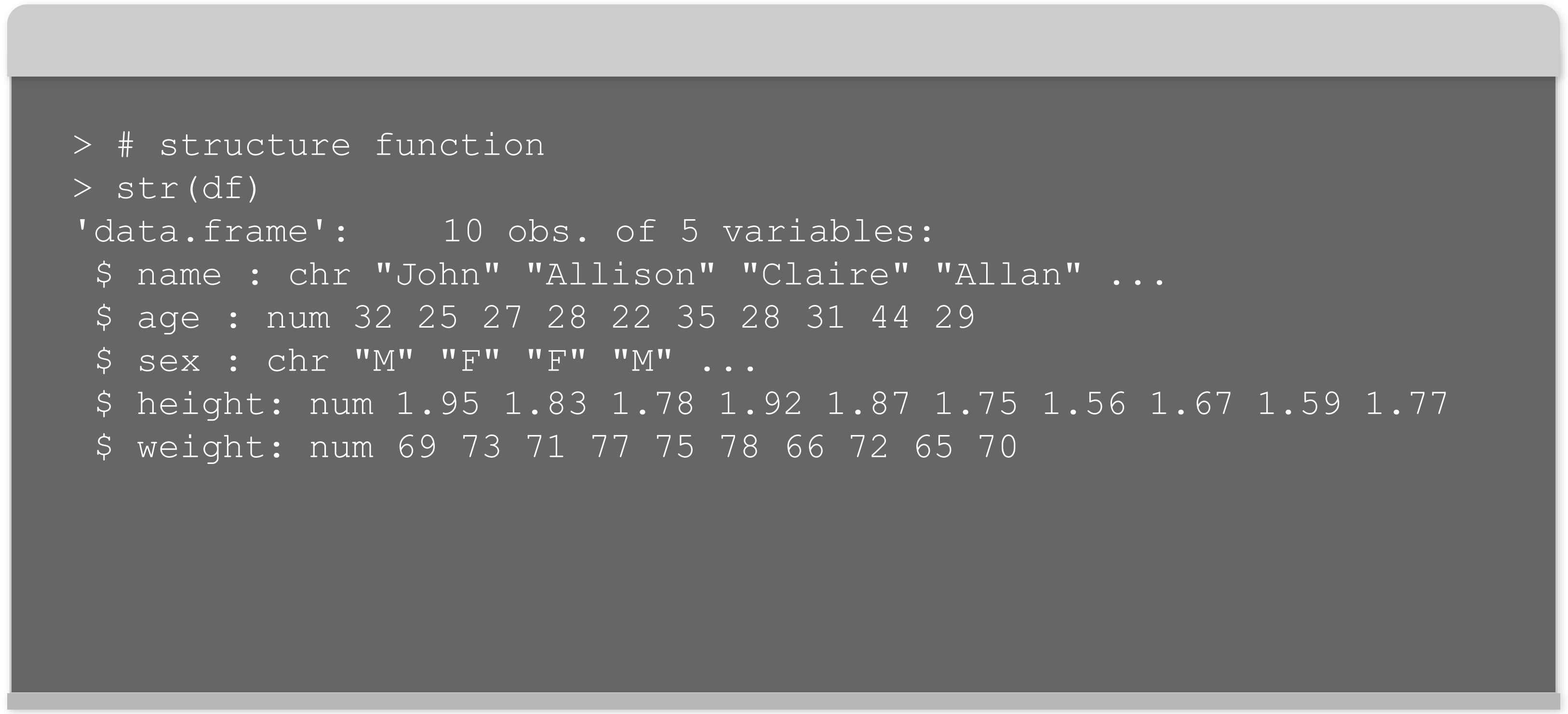
```
> # display top 6 rows of the dataframe df
> head(df)
 name age sex height weight
1 John 32 M 1.95 69
2 Allison 25 F 1.83 73
3 Claire 27 F 1.78 71
4 Allan 28 M 1.92 77
5 Debra 22 F 1.87 75
6 Jack 35 M 1.75 78
```

tail(df)

```
> # display bottom 6 rows of the dataframe df
> tail(df)
 name age sex height weight
5 Debra 22 F 1.87 75
6 Jack 35 M 1.75 78
7 Reed 28 F 1.56 66
8 Arron 31 M 1.67 72
9 Paul 44 M 1.59 65
10 Stephanie 29 F 1.77 70
```

Structural Information

The str(x) function displays information about the data columns of the dataframe along with their respective class. Example:



```
> # structure function
> str(df)
'data.frame':    10 obs. of 5 variables:
 $ name : chr "John" "Allison" "Claire" "Allan" ...
 $ age : num 32 25 27 28 22 35 28 31 44 29
 $ sex : chr "M" "F" "F" "M" ...
 $ height: num 1.95 1.83 1.78 1.92 1.87 1.75 1.56 1.67 1.59 1.77
 $ weight: num 69 73 71 77 75 78 66 72 65 70
```

Exporting Data in R

Exporting Data in R

The data can be exported in three file formats, such as:

TXT — Text file: write.table() and write_tsv()

CSV — CSV file: write.table(), write.csv(), write_csv(), and write.csv2()

XLS — Excel file: write.xlsx()

Exporting Data in Text File

The write.table() and write_tsv() functions export a data frame or a matrix file to a tab-separated value text file.

write.table()

```
write.table(x, file, append = FALSE, sep = " ",
dec = ".", row.names = TRUE, col.names = TRUE)
```

readr library: write_tsv()

```
write_tsv(file, path)
```

Exporting Data in Text File

The attributes specified for exporting the data are listed below:

Attributes	Description
x	Specify a data frame or matrix
file	Specify the name of the resultant file
sep	Denotes the field separator string (\t)
dec	Denotes the string to be used as a decimal separator, default (.)
row.names	Specify a logical value which can be a row name of x or a character vector
col.names	Specify a logical value which can be a column name of x or a character vector

Exporting Data in Text File

An example to export data in a text file is shown below:

write.table()

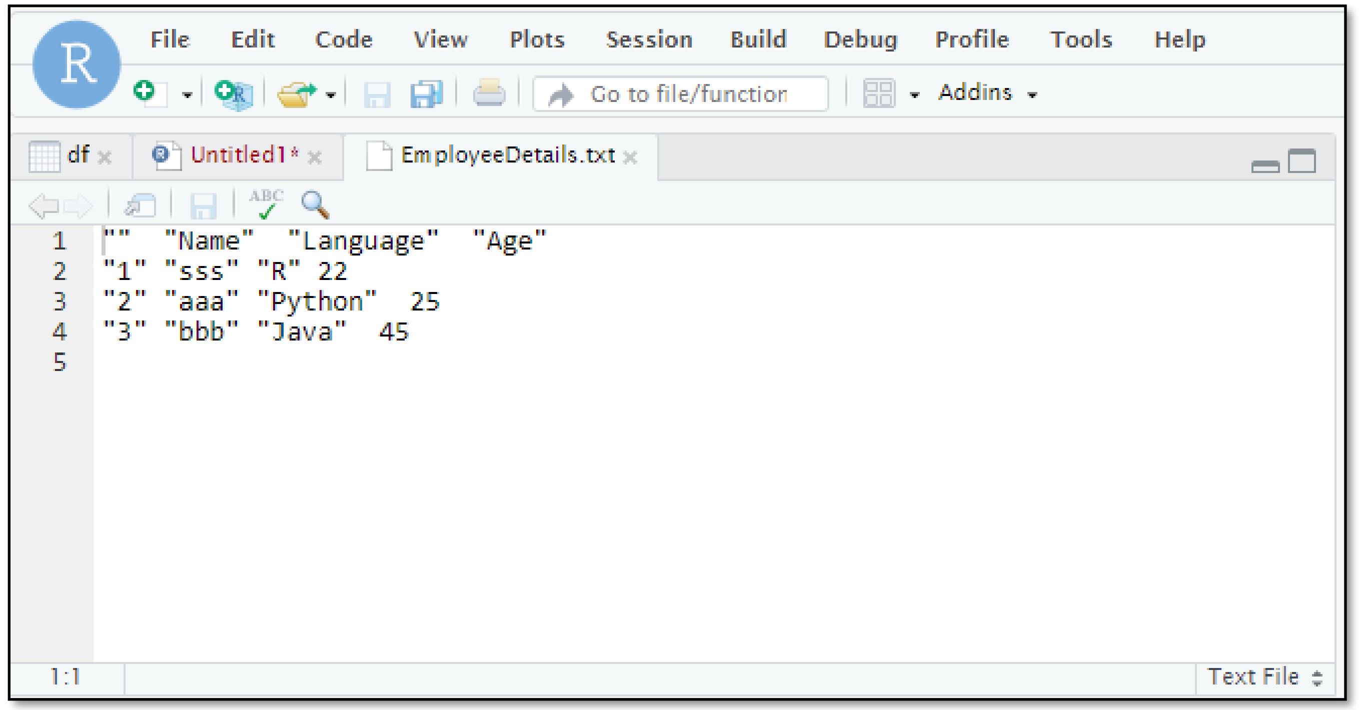
```
# Creating a dataframe
df = data.frame(
"Name" = c("Jason", "Jacob, "Fin"),
"Language" = c("Developer", "CEO", "Manager"),
"Age" = c(26, 35, 40)
)

# Export a data frame to a text file using
write.table()
write.table(df,
file = "EmployeeDetails.txt",
sep = "\t",
row.names = TRUE,
col.names = NA)
```

readr library:
write_tsv()

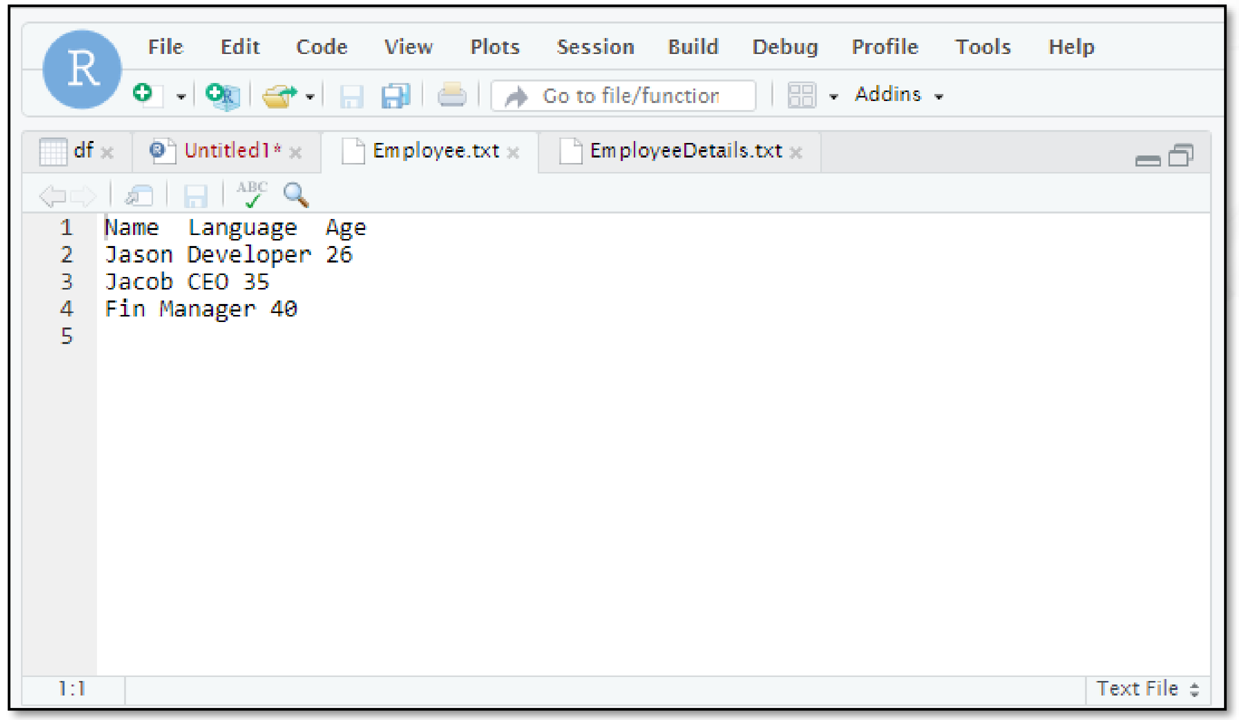
```
# Importing readr library
library(readr)

# Creating a dataframe
df = data.frame(
 "Name" = c("Jason", "Jacob", "Fin"),
 "Language" = c("Developer", "CEO", "Manager"),
 "Age" = c(26, 35, 40)
)

# Export a data frame using write_tsv()
write_tsv(df, path = "Employee.txt")
```

Exporting Data in Text File

The final output of write.table() and write_tsv() are:



```
""   "Name"   "Language"   "Age"
"1" "sss" "R" 22
"2" "aaa" "Python"  25
"3" "bbb" "Java"  45
```



```
Name  Language  Age
Jason Developer 26
Jacob CEO 35
Fin Manager 40
```

Exporting Data in CSV File

The write.table(), write.csv(), write_csv(), and write.csv2() functions export a data frame or a matrix file to a comma-separated value CSV file.

write.table()

```
write.table(x, file, append = FALSE, sep = ” “,
dec = “.”, row.names = TRUE, col.names = TRUE)
```

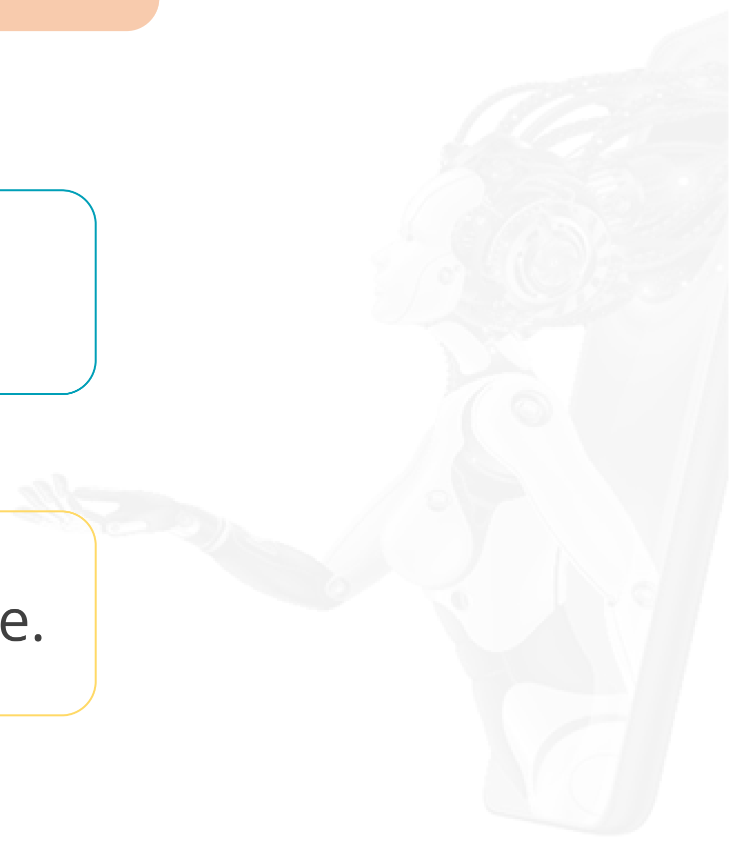
write.csv(), write_csv(), write.csv2()

```
write_csv(file, path)
write.csv(file, path)
Write.csv2(file,path)
```

Exporting Data in CSV File

Each function uses a different character for decimal points and separators.

write.csv()	It uses (.) for the decimal point and a comma (,) for the separator.
write_csv	It uses comma-separated (,) values using the readr package.
write.csv2()	It uses a comma (,) for the decimal point and a semicolon (;) for the separator.



Exporting Data in CSV File

An example to export data in a CSV file is shown below:

write.table()

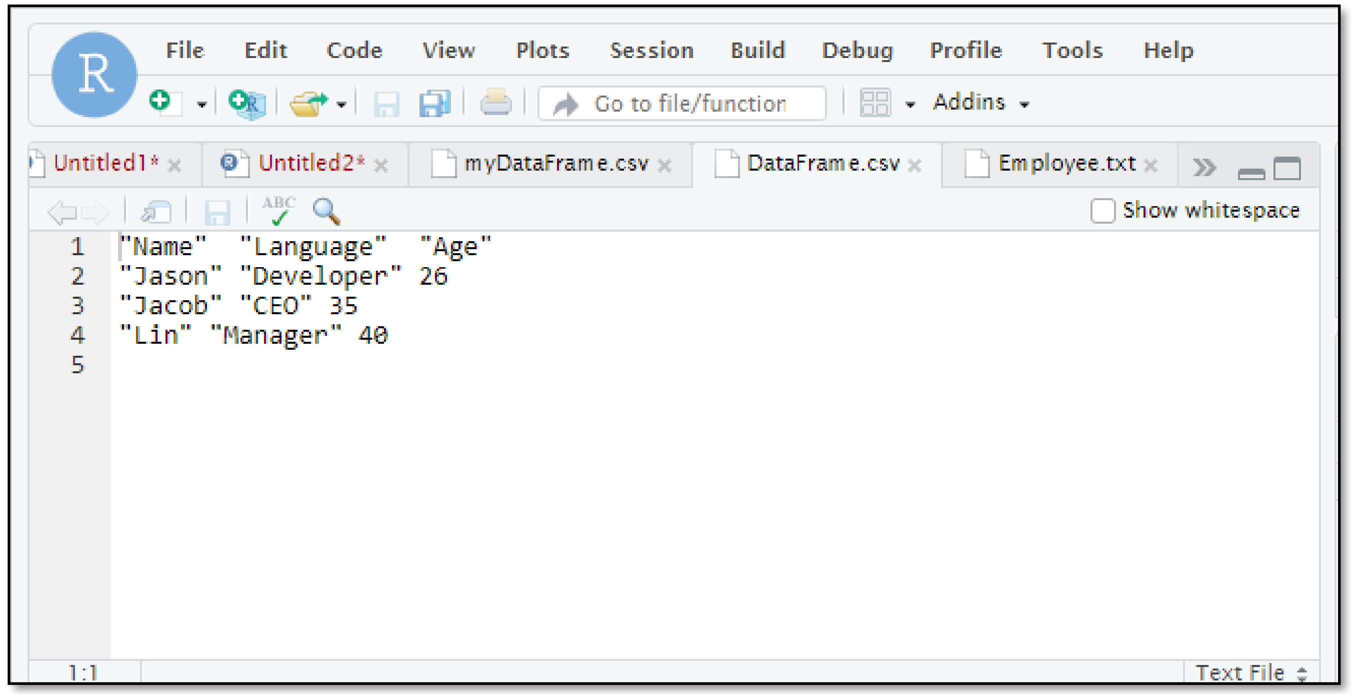
```
# Exporting data from R

# Creating a dataframe
df = data.frame(
 "Name" = c("Jason", "Jacob", "Lin"),
 "Language" = c("Developer", "CEO", "Manager"),
 "Age" = c(26, 35, 40)
)

# Export a data frame to a csv file using
write.table()
write.table(df,
 file = "DataFrame.csv",
 sep = "\t",
 row.names = FALSE,
)
```

write.csv(), write_csv(), write.csv2()

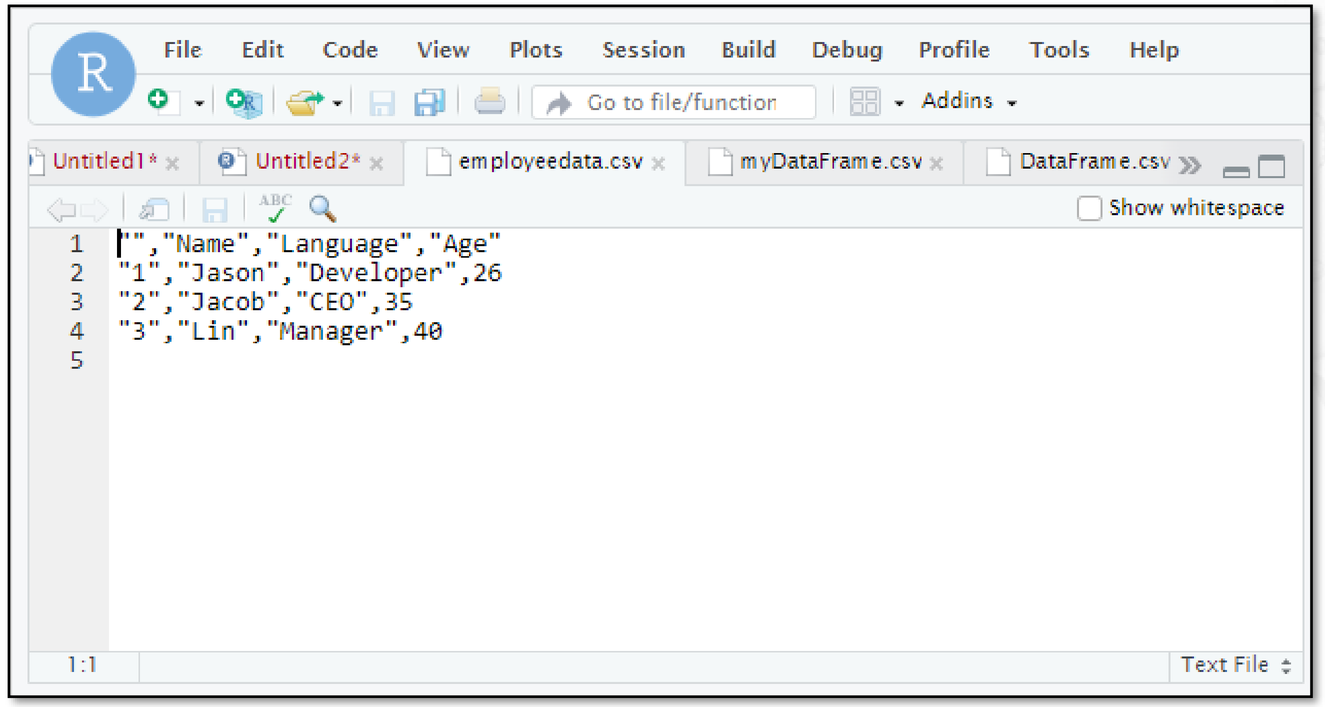
```
# Creating a dataframe
df = data.frame(
 "Name" = c("Jason", "Jacob", "Lin"),
 "Language" = c("Developer", "CEO", "Manager"),
 "Age" = c(26, 35, 40)
)

# Export a data frame to a csv file using write.csv()
write.csv(df, file = "employeedata.csv")

# Export a data frame to a csv file using
write.csv2()
write.csv2(df, file = "emp.csv")

# Export a data frame to a csv file using write_csv()
library(readr)
write_csv(df, path = "empdetails.csv")
```

Exporting Data in CSV File

The final output of write.table() and write.csv() are shown below:



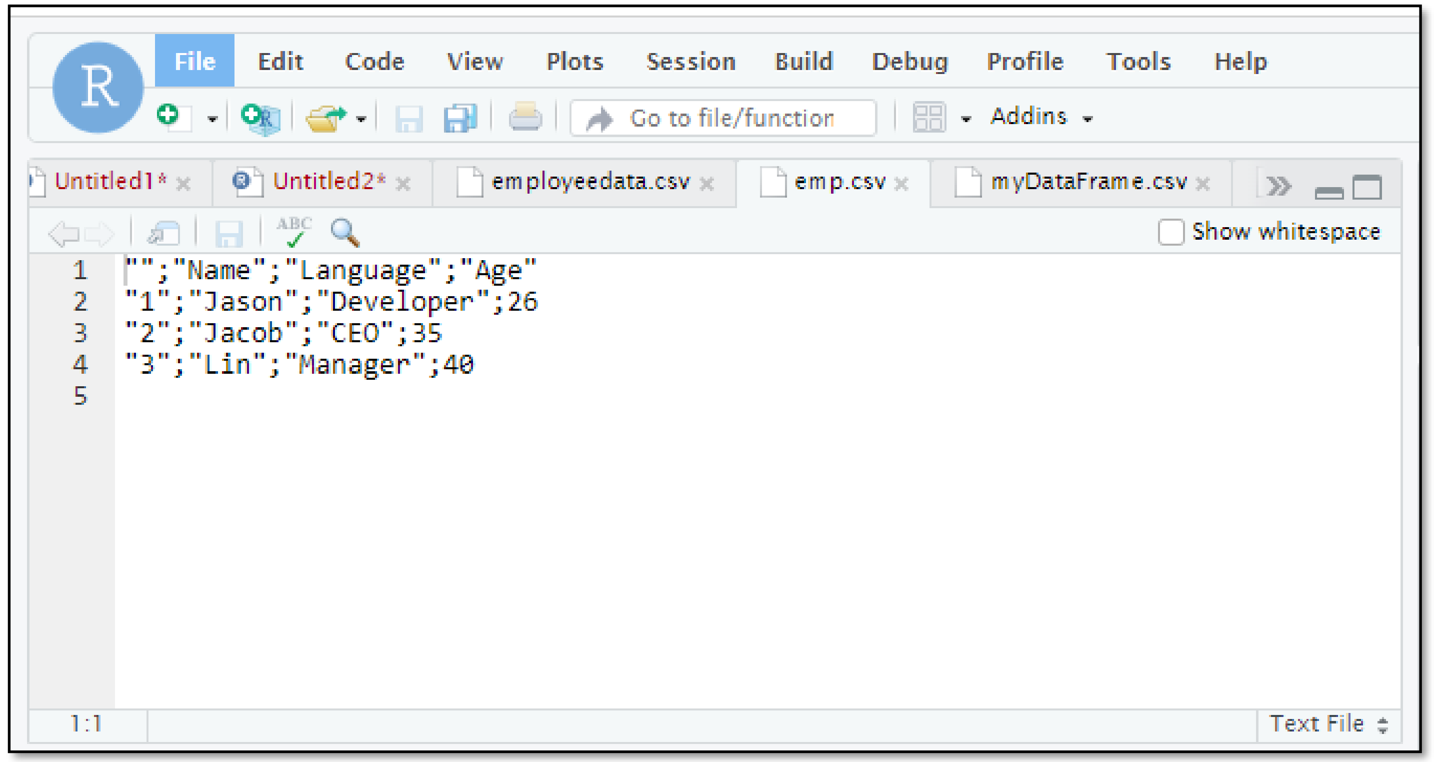
```
"Name"  "Language"  "Age"
"Jason" "Developer" 26
"Jacob" "CEO" 35
"Lin" "Manager" 40
```



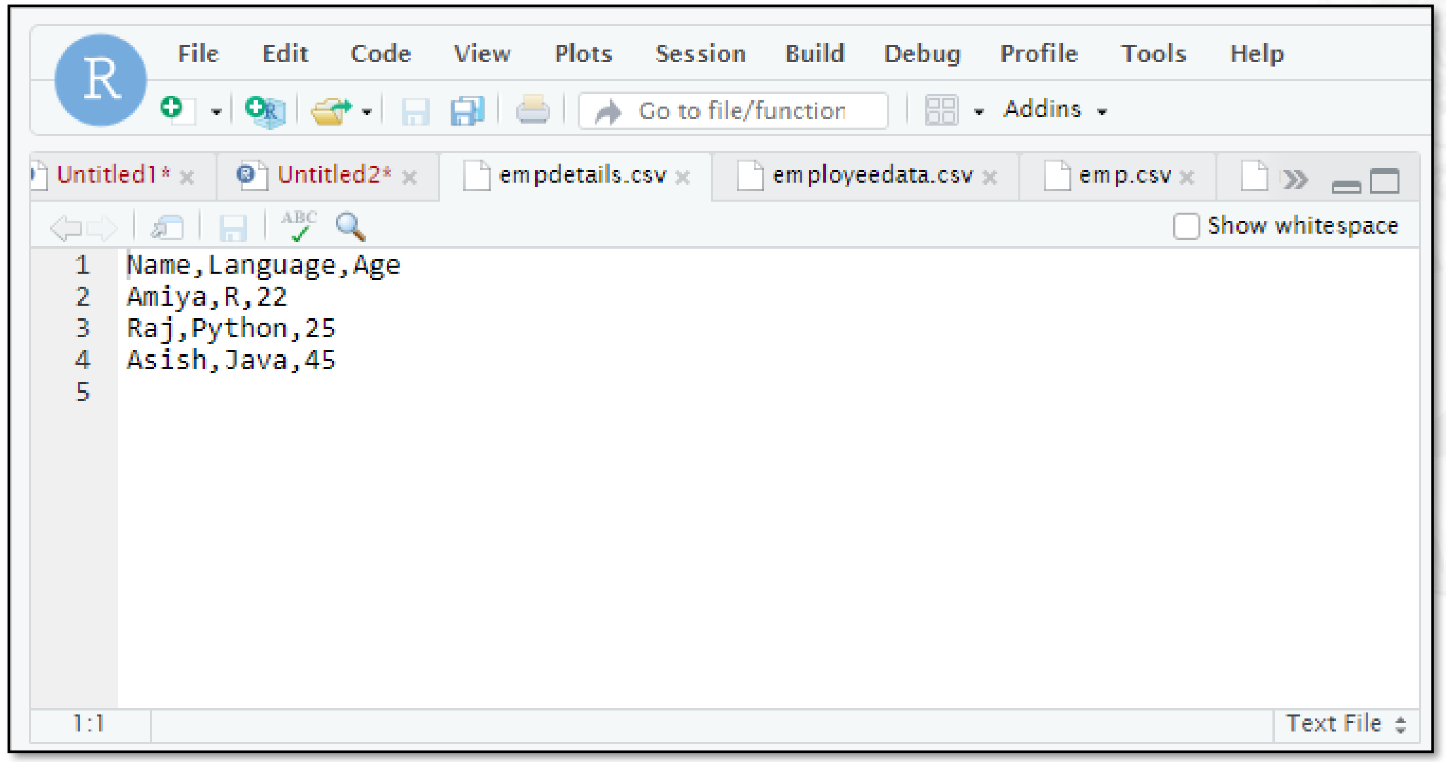
```
"","Name","Language","Age"
"1","Jason","Developer",26
"2","Jacob","CEO",35
"3","Lin","Manager",40
```

Exporting Data in CSV File

The final output of write.csv2() and write_csv() are shown below:



```
"";"Name";"Language";"Age"
"1";"Jason";"Developer";26
"2";"Jacob";"CEO";35
"3";"Lin";"Manager";40
```



```
Name,Language,Age
Amiya,R,22
Raj,Python,25
Asish,Java,45
```

Exporting Data in Excel File

The write.xlsx() function export a data frame or a matrix file to an Excel file.

Syntax: write.xlsx()

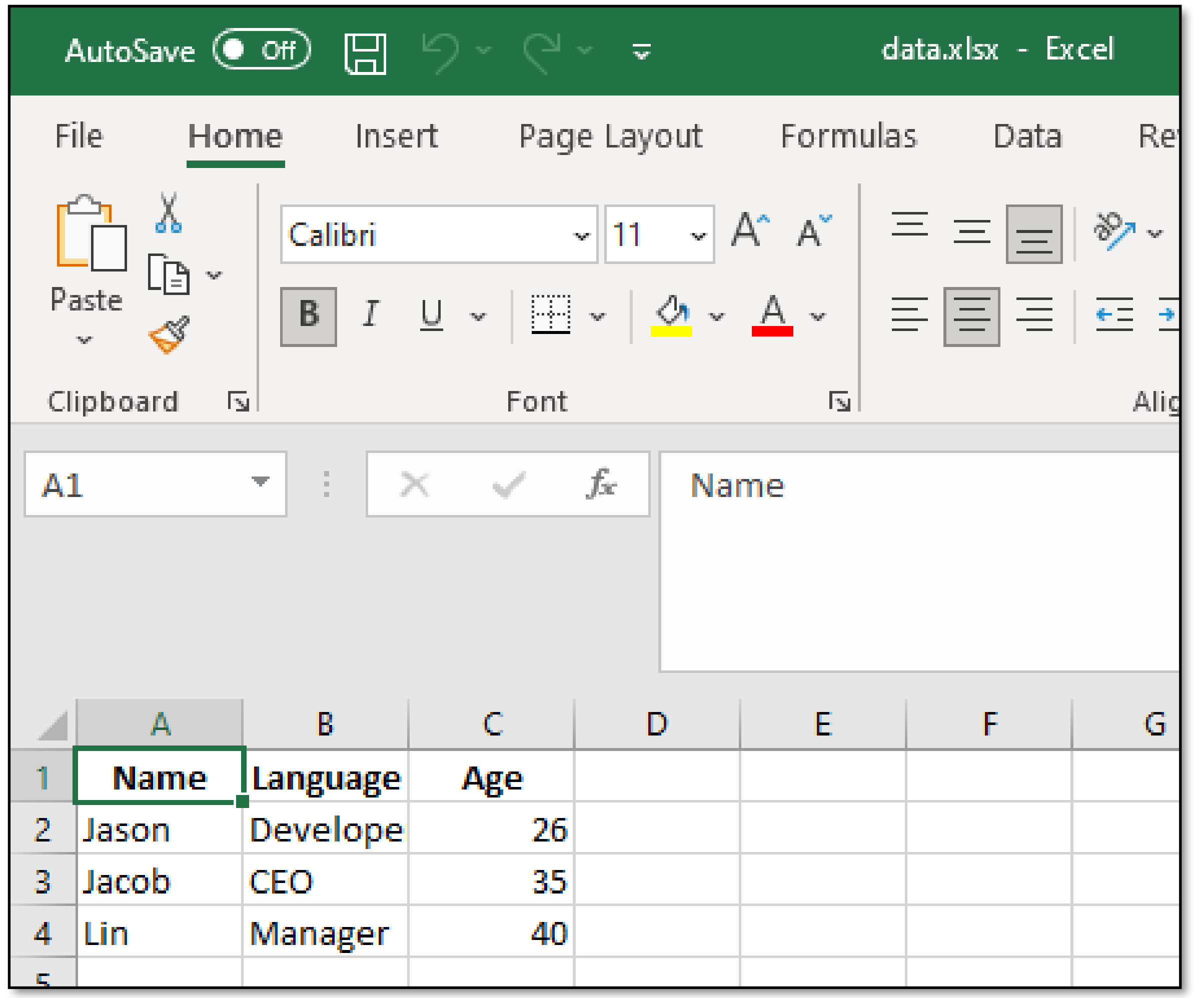
```
library(xlsx)
write.xlsx(file,path)
```

Example: write.xlsx()

```
# Creating a dataframe
library(xlsx)
df = data.frame(
 "Name" = c("Jason", "Jacob", "Lin"),
 "Language" = c("Developer", "CEO", "Manager"),
 "Age" = c(26, 35, 40)
)

write_xlsx(df, path = "data.xlsx")
```

Exporting Data in Excel File

The final output of write.xlsx() is shown below:

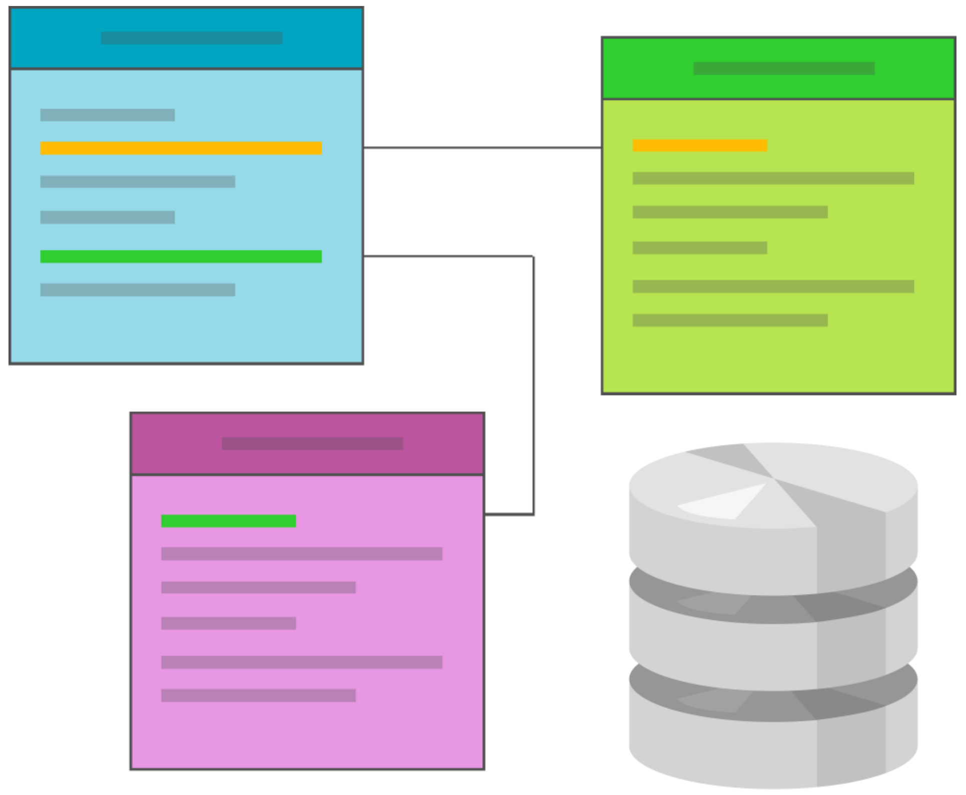


	A	B	C	D	E	F	G
1	Name	Language	Age				
2	Jason	Develope	26				
3	Jacob	CEO	35				
4	Lin	Manager	40				

Database Connectivity in R

Database Connectivity

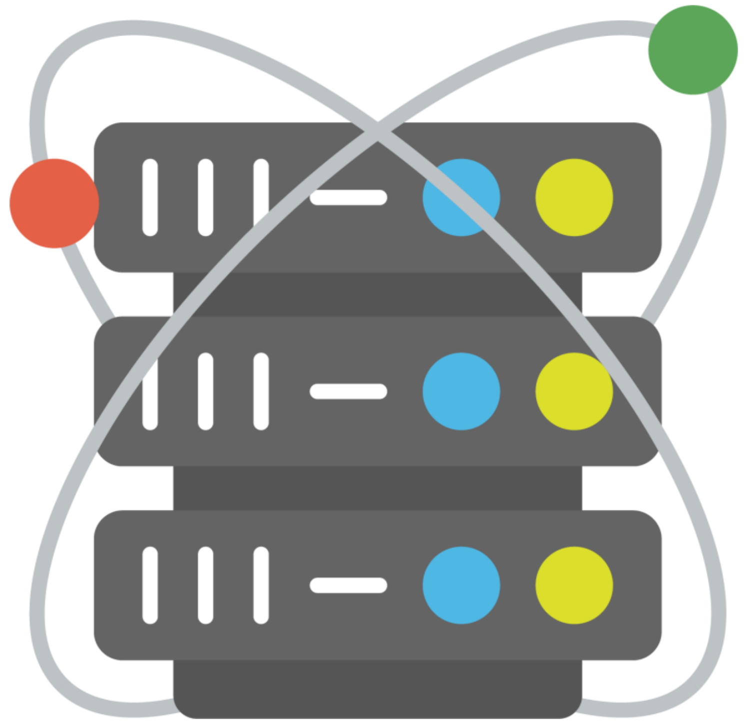
It is used to access relational databases and fetch records as data frames.



The imported data can be used to analyze and manipulate using powerful packages and functions in R.

Need for Database Connectivity

Data analysts can use it to analyze the data that IT professionals have stored.



Database Connectivity

There are two ways to use a database in R, such as:

The in-built packages in R have databases that can perform operations in R.

The phpMyAdmin local host or MySQL tool helps to create a database and perform operations in R.

Software Packages

Database Interface (DBI) and Open Database Connectivity (ODBC) are two main packages used for connecting to and working with databases in R.

The ODBC package provides a set of drivers for connecting to databases, such as SQL servers, Amazon Web Services Redshift, and Google Cloud databases.

The DBI package consists of functions for interacting with the database.

Software Packages

There are other packages that can be used to create database connectivity within R and MySQL, such as:

The RMySQL package is used to work with SQL databases.

The RSQLite package includes SQLite, which allows using a database from R.

The tidyverse is a collection of R packages used to perform and interact with data in data science.

Software Packages

The syntax and example for installing the software packages are shown below:

Syntax

```
install.packages(packagename)
```

Example

```
install.packages("odbc")
install.packages("DBI")
install.packages("RMySQL")
install.packages("RSQLite")
install.packages("tidyverse")
```

Database Connectivity

The dbConnect() function creates a connection with the database.

It returns a database connection profile, which is stored in an R object called con.

The con object can be used only after the connection is created to access the database with a variety of functions.

Database Connectivity

The following syntax and example help to create database connectivity using the databases present in the RSQLite package.

Syntax

```
con <- DBI::dbConnect(drv = odbc::odbc(),
 Driver = "driver_name",
 Server = "server_url",
 Database = "database_name",
 user = "user", #optional
 password = "password") #optional
```

Example

```
library(odbc)
library(DBI)

library(RMySQL) #Generate and Process 'SQL'
Queries in R
library(RSQLite) #Can create an in-memory SQL
database
library(tidyverse)

#Build the placeholder
con <- dbConnect(drv = RSQLite::SQLite(),
 dbname = ":memory:")

dbListTables(con))
```

Subsetting Dataframes

Subsetting Dataframes

A subset can be created by using square brackets to extract specific rows and columns.

A dollar operator can be used to extract a column from the dataframe by the name of the column.

A subset can also be created using conditional filtering.

Extract Specific Rows and Columns

The example here shows how to extract a specific row and column from the ***cars.csv*** data imported as ***data***.

```
> # extract a particular row
> data[25 , ]
 Make Model Type Origin DriveTrain MSRP Invoice EngineSize Cylinders Horsepower
25 Hyundai XG350 L 4dr Sedan Asia Front NA 23486 3.5 6 194
 MPG_City MPG_Highway Weight Wheelbase Length
25 17 26 3651 108 192
> # extract a particular column by index
> data[ , 5]
 [1] All Front Front Rear Front Rear Front All Front Rear All All
[13] Rear Front Front Front Rear Rear Rear Front Rear Front Front All
[25] Front Front Front Front All Front Rear Front Front Front All Front
[37] Rear Rear All All Front All Front Front Front Front Front Front
[49] Front Rear
 [ reached getOption("max.print") -- omitted 378 entries ]
Levels: All Front Rear
```

Extract Specific Rows and Columns

The df[row, columns] format can be used to extract multiple rows and columns.

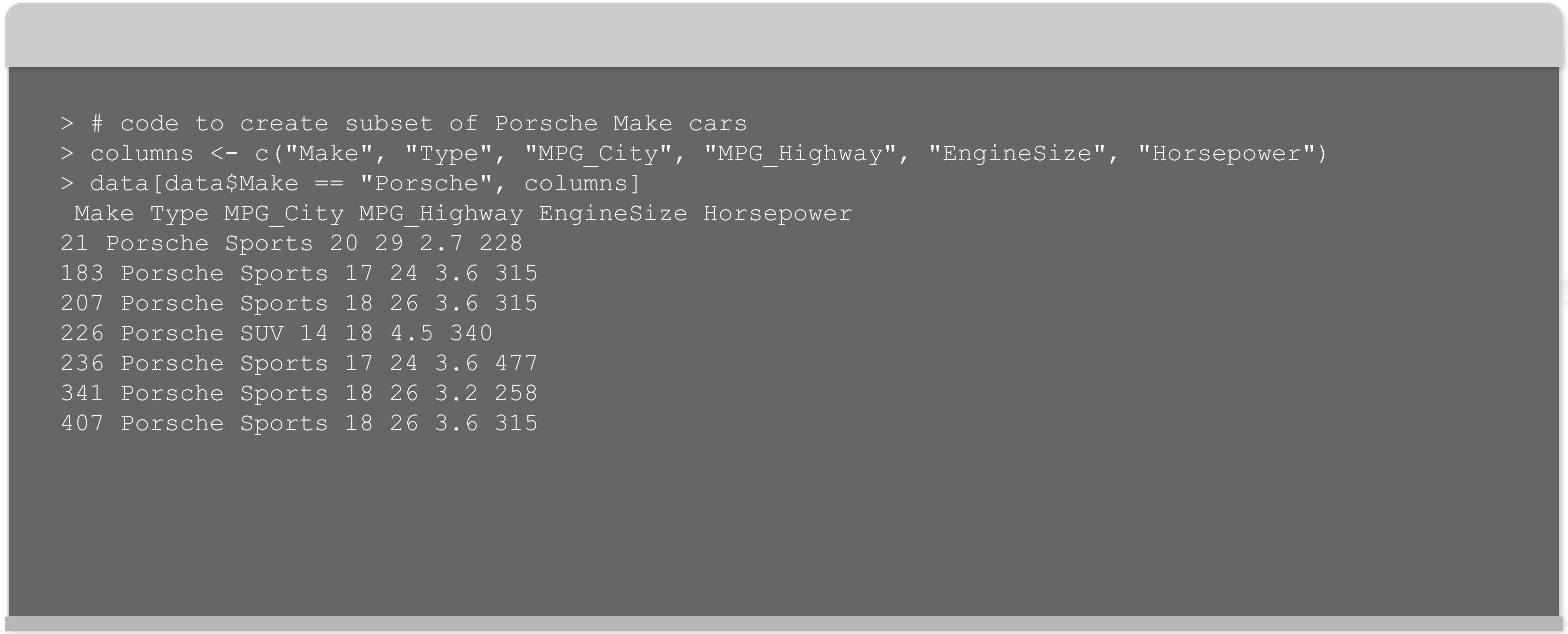
Using ***cars.csv*** for subsetting:

```
> # extract first 5 rows and first 5 columns from the data frame
> data[1:5, 1:5]
 Make Model Type Origin DriveTrain
1 Subaru Forester X Wagon All
2 Toyota Camry Solara SE V6 2dr Sedan Asia Front
3 Suzuki Aerio LX 4dr Sedan Asia Front
4 Dodge Dakota Club Cab Truck USA Rear
5 Mazda Mazda3 s 4dr Sedan Asia Front
> # a single column can be extracted as a vector using dollar operator.
> data$Make
 [1] Subaru Toyota Suzuki Dodge Mazda
 [6] Infiniti Pontiac GMC Chevrolet Mercedes-Benz
[11] BMW Subaru
 [ reached getOption("max.print") -- omitted 416 entries ]
38 Levels: Acura Audi BMW Buick Cadillac Chevrolet Chrysler Dodge ... Volvo
```

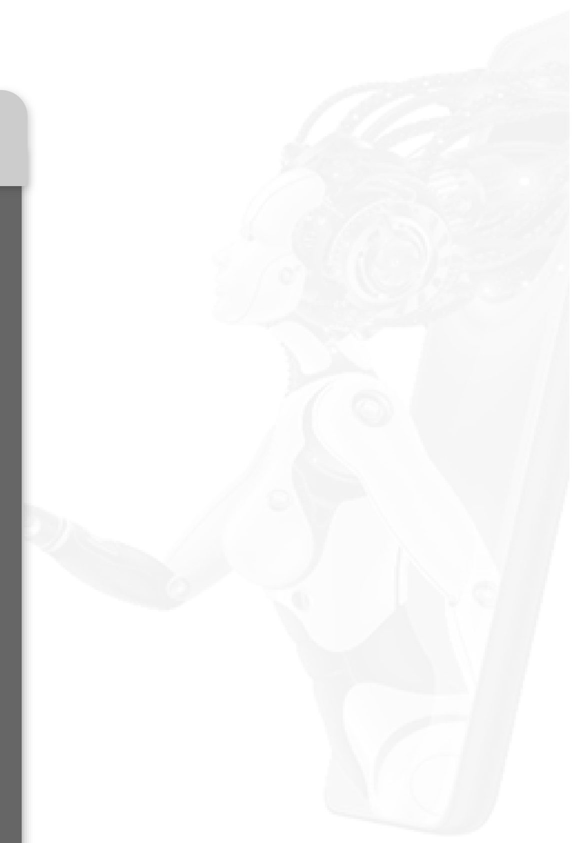
Conditional Filtering

Subsets of data can be extracted based on conditions.

For example, the data for Porsche cars is extracted here:



```
> # code to create subset of Porsche Make cars
> columns <- c("Make", "Type", "MPG_City", "MPG_Highway", "EngineSize", "Horsepower")
> data[data$Make == "Porsche", columns]
 Make Type MPG_City MPG_Highway EngineSize Horsepower
21 Porsche Sports 20 29 2.7 228
183 Porsche Sports 17 24 3.6 315
207 Porsche Sports 18 26 3.6 315
226 Porsche SUV 14 18 4.5 340
236 Porsche Sports 17 24 3.6 477
341 Porsche Sports 18 26 3.2 258
407 Porsche Sports 18 26 3.6 315
```



Conditional Filtering: Multiple Conditions

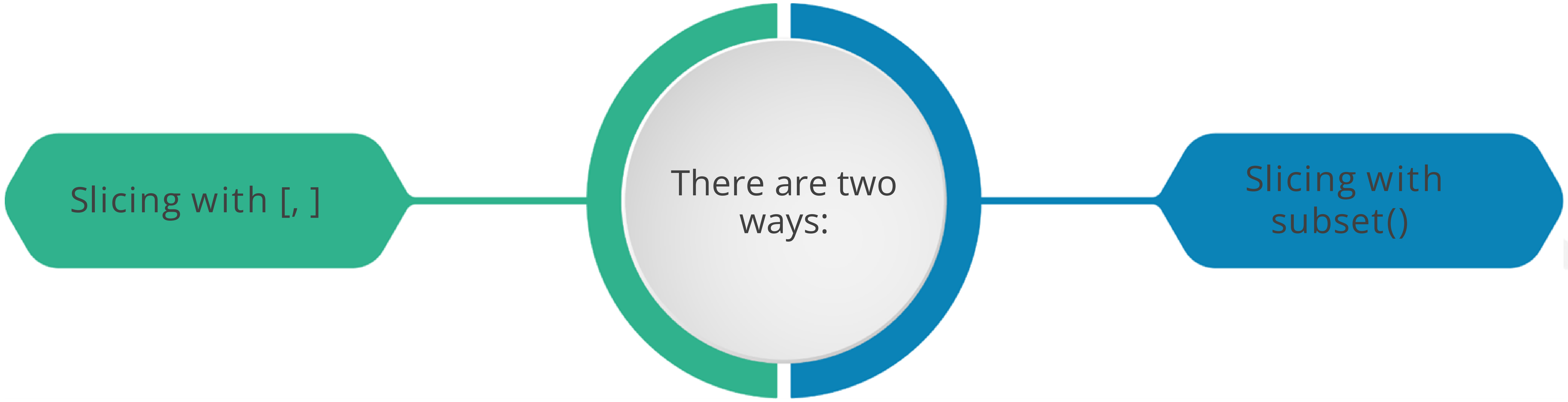
A dataframe can be filtered for multiple conditions using the element-wise logical operators of & and |.

For example, a subset of data for Sports cars manufactured by Porsche is created here:

```
> # code to create subset for Sports cars manufactured by Porsche
> columns <- c("Make", "Type", "MPG_City", "MPG_Highway", "EngineSize", "Horsepower")
> data[data$Make == "Porsche" & data$Type =="Sports", columns]
 Make Type MPG_City MPG_Highway EngineSize Horsepower
21 Porsche Sports 20 29 2.7 228
183 Porsche Sports 17 24 3.6 315
207 Porsche Sports 18 26 3.6 315
236 Porsche Sports 17 24 3.6 477
341 Porsche Sports 18 26 3.2 258
407 Porsche Sports 18 26 3.6 315
```

Slicing and Dicing Data

It is the process of selecting specific rows and columns of data based on some criteria.



Slicing helps to filter rows, whereas dicing selects a set of columns from the data.

Slicing and Dicing [,]

It represents two index vectors such as rows and columns.

Syntax

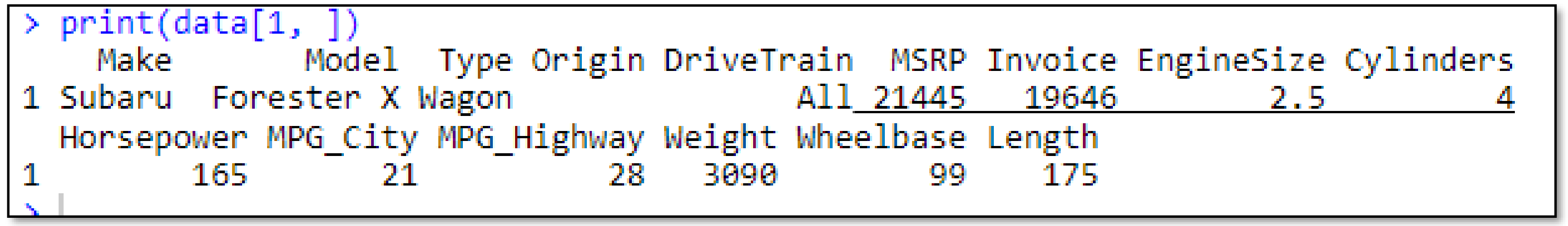
```
Dataframe name[rows, columns]
```

Example

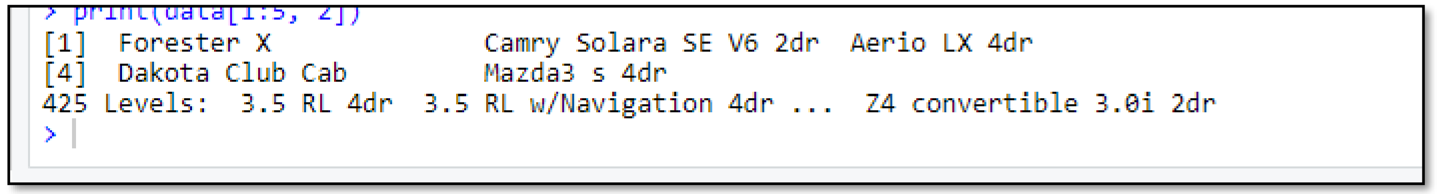
```
data <- read.csv('cars.csv’, header =
TRUE, stringsAsFactors = TRUE)
# Return row 1
print(data[1, ])

# Return column 5
print(data[, 5])

# Rows 1:5 and column 2
print(data[1:5, 2])
```

Slicing and Dicing [,]

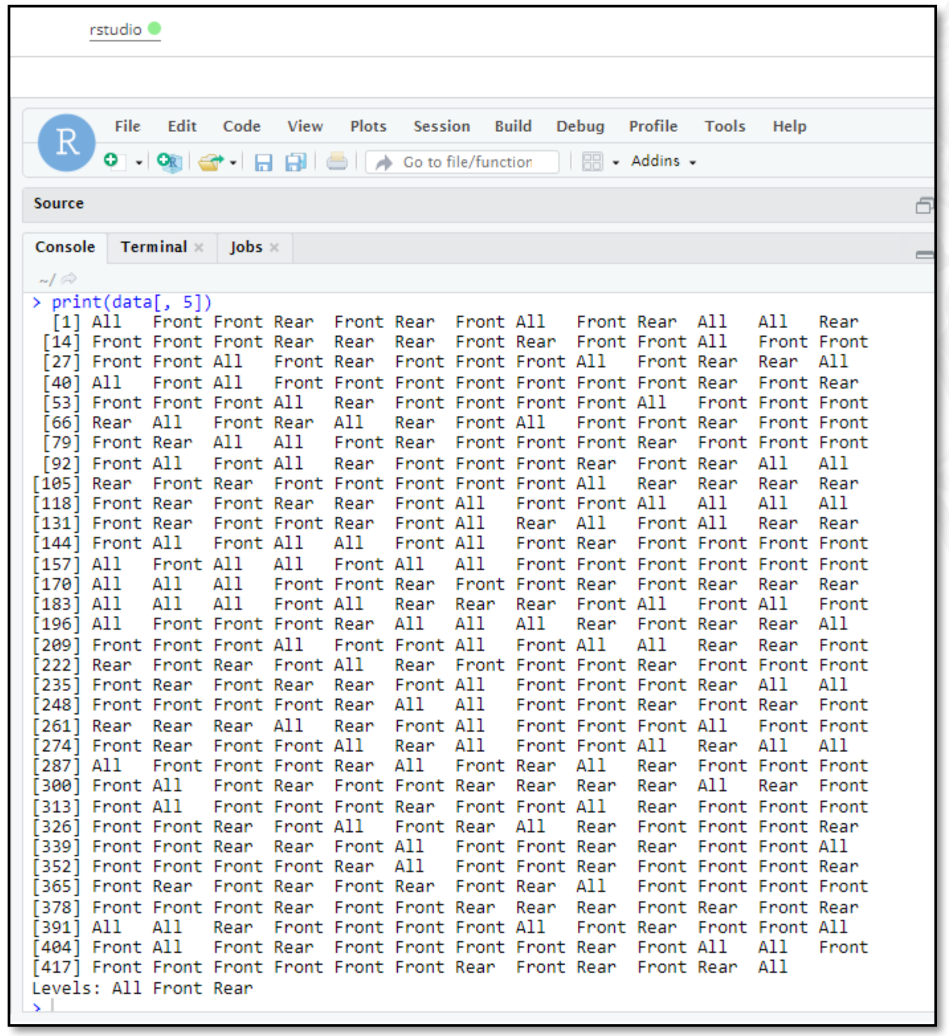
The final output is shown below:



```
> print(data[1, ])
    Make        Model   Type Origin DriveTrain  MSRP Invoice EngineSize Cylinders
1 Subaru  Forester X Wagon                  All 21445   19646        2.5         4
  Horsepower MPG_City MPG_Highway Weight Wheelbase Length
1        165       21          28   3090        99    175
> 
```



```
> print(data[1:5, 2])
[1]  Forester X           Camry Solara SE V6 2dr   Aerio LX 4dr
[4]  Dakota Club Cab      Mazda3 s 4dr
425 Levels:  3.5 RL 4dr  3.5 RL w/Navigation 4dr ...  Z4 convertible 3.0i 2dr
> 
```



```
> print(data[, 5])
  [1] All   Front Front Rear  Front Rear  Front All   Front Rear  All   All   Rear 
 [14] Front Front Front Rear  Rear  Rear  Front Rear  Front Front All   Front Front
 [27] Front Front All   Front Rear  Front Front Front All   Front Rear  Rear  All  
 [40] All   Front All   Front Front Front Front Front Front Front Rear  Front Rear 
 [53] Front Front Front All   Rear  Front Front Front Front All   Front Front Front
 [66] Rear  All   Front Rear  All   Rear  Front All   Front Front Rear  Front Front
 [79] Front Rear  All   All   Front Rear  Front Front Front Rear  Front Front Front
 [92] Front All   Front All   Rear  Front Front Front Rear  Front Rear  All   All  
[105] Rear  Front Rear  Front Front Front Front Front All   Rear  Rear  Rear  Rear 
[118] Front Rear  Front Rear  Rear  Front All   Front Front All   All   All   All  
[131] Front Rear  Front Front Rear  Front All   Rear  All   Front All   Rear  Rear 
[144] Front All   Front All   All   Front All   Front Rear  Front Front Front Front
[157] All   Front All   All   Front All   All   Front Front Front Front Front Front
[170] All   All   All   Front Front Rear  Front Front Rear  Front Rear  Rear  Rear 
[183] All   All   All   Front All   Rear  Rear  Rear  Front All   Front All   Front
[196] All   Front Front Front Rear  All   All   All   Rear  Front Rear  Rear  All  
[209] Front Front Front All   Front Front All   Front All   All   Rear  Rear  Front
[222] Rear  Front Rear  Front All   Rear  Front Front Front Rear  Front Front Front
[235] Front Rear  Front Rear  Rear  Front All   Front Front Front Rear  All   All  
[248] Front Front Front Front Rear  All   All   Front Front Rear  Front Rear  Front
[261] Rear  Rear  Rear  All   Rear  Front All   Front Front Front All   Front Front
[274] Front Rear  Front Front All   Rear  All   Front Front All   Rear  All   All  
[287] All   Front Front Front Rear  All   Front Rear  All   Rear  Front Front Front
[300] Front All   Front Rear  Front Front Rear  Rear  Rear  Rear  All   Rear  Front
[313] Front All   Front Front Front Rear  Front Front All   Rear  Front Front Front
[326] Front Front Rear  Front All   Front Rear  All   Rear  Front Front Front Rear 
[339] Front Front Rear  Rear  Front All   Front Front Rear  Rear  Front Front All  
[352] Front Front Front Front Rear  All   Front Front Rear  Front Front Front Rear 
[365] Front Rear  Front Rear  Front Rear  Front Rear  All   Front Front Front Front
[378] Front Front Front Rear  Front Front Rear  Rear  Rear  Front Rear  Front Rear 
[391] All   All   Rear  Front Front Front Front All   Front Rear  Front Front All  
[404] Front All   Front Rear  Front Front Front Front Rear  Front All   All   Front
[417] Front Front Front Front Front Front Rear  Front Rear  Front Rear  All  
Levels: All Front Rear
> 
```

Subset() Function

The subset() function can also be used to extract subsets of data if given conditions are met.

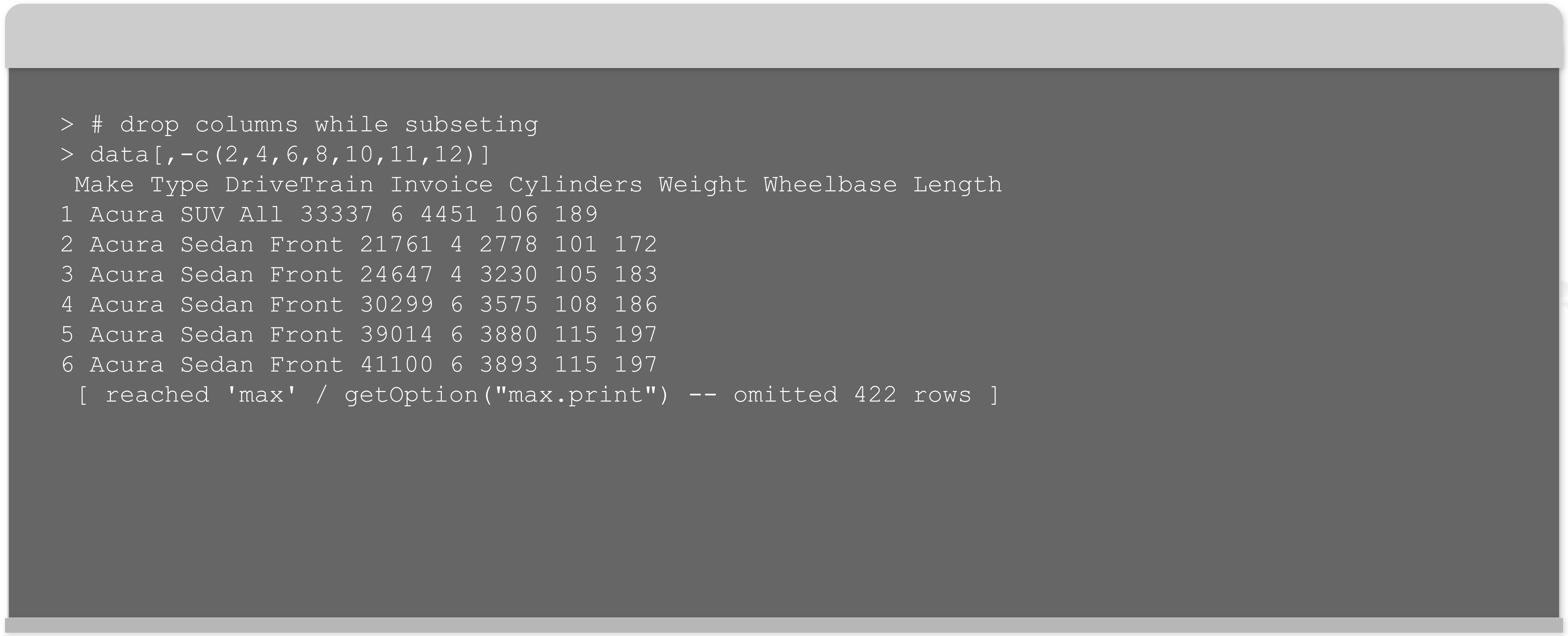
For example, a subset of data where “MSRP” is greater than $100,000 is shown:

```
> # fetch data for observations where MSRP is greater than 100000
> subset(data, subset =MSRP > 100000,
+ select = c("Make", "Model", "Type","MSRP", "MPG_City", "Horsepower"))
 Make Model Type MSRP MPG_City Horsepower
236 Porsche 911 GT2 2dr Sports 192465 17 477
238 Mercedes-Benz SL600 convertible 2dr Sports 126670 13 493
348 Mercedes-Benz SL55 AMG 2dr Sports 121770 14 493
```

Dropping Variables

Existing variables can be dropped while creating subsets of data.

It can be achieved by using negative index.



```
> # drop columns while subseting
> data[,-c(2,4,6,8,10,11,12)]
 Make Type DriveTrain Invoice Cylinders Weight Wheelbase Length
1 Acura SUV All 33337 6 4451 106 189
2 Acura Sedan Front 21761 4 2778 101 172
3 Acura Sedan Front 24647 4 3230 105 183
4 Acura Sedan Front 30299 6 3575 108 186
5 Acura Sedan Front 39014 6 3880 115 197
6 Acura Sedan Front 41100 6 3893 115 197
 [ reached 'max' / getOption("max.print") -- omitted 422 rows ]
```

Creating New Variables

Adding Variables

New variables can be added to dataframes as:

- ✓ A new field vector
- ✓ A recoding of existing column
- ✓ A calculated field from existing columns

New Variable From a New Vector

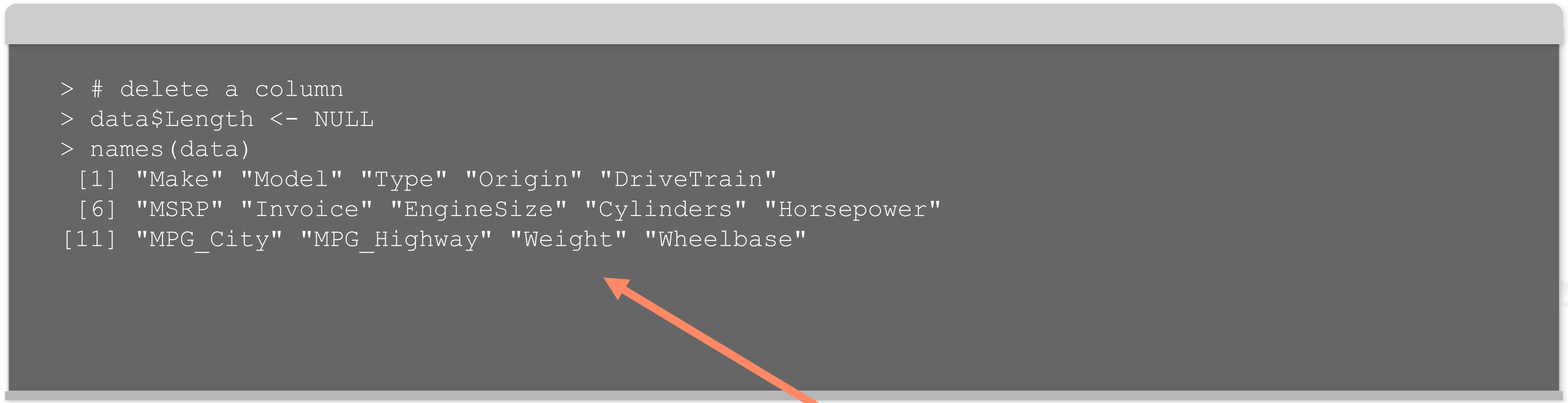
Consider the example of creating a new column in “data”.

```
> # reading the data from a new file
> year <- read_excel('year.xlsx')
> year$Year
 [1] 2015 2011 2012 2014 2013 2013 2013 2014 2014 2015 2011 2015 2012 2013 2013 2011 2015 2012
[19] 2012 2015 2014 2011 2012 2011 2012 2014 2015 2012 2011 2011 2015 2012 2011 2011 2015 2014
[37] 2014 2013 2012 2011 2012 2013 2014 2011 2013 2015 2014 2013 2015 2012
 [ reached getOption("max.print") -- omitted 378 entries ]
> # assigning data to a new column
> data$Year <- year$Year
> # new column as seen in the names attribute
> names(data)
 [1] "Make" "Model" "Type" "Origin" "DriveTrain" "MSRP"
 [7] "Invoice" "EngineSize" "Cylinders" "Horsepower" "MPG_City" "MPG_Highway"
[13] "Weight" "Wheelbase" "Length" "Year"
```

The year$Year is a vector.

Delete a Column

To delete a column, it can be assigned to NULL.



```
> # delete a column
> data$Length <- NULL
> names(data)
 [1] "Make" "Model" "Type" "Origin" "DriveTrain"
 [6] "MSRP" "Invoice" "EngineSize" "Cylinders" "Horsepower"
[11] "MPG_City" "MPG_Highway" "Weight" "Wheelbase"
```

The “Length” column no more exists in the data.

Recoding Variable

Duration: 5 minutes

Problem Statement: Consider the data in the cars.csv file and recode the length variable as size codes based on the below criterion.

Length Range	Size Code
140 - 190	Small
191 - 200	Mid
201 - 240	Large

Note: Please download the data set and the solution document from the Course Resources section and follow the steps given in the document

Sorting Data

Sorting

Sorting needs to be done to arrange data in a meaningful order for more effective analysis.

The order(x,) function is used for sorting of dataframes:

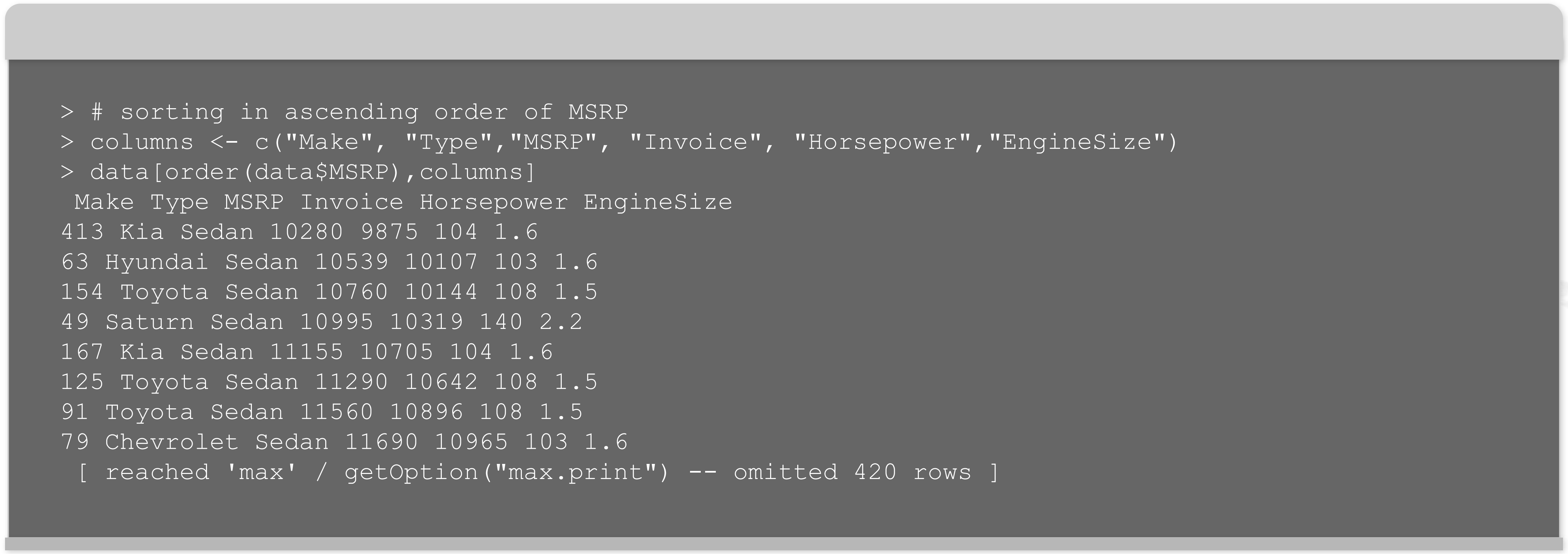
Syntax : order(x, decreasing = FALSE)

The function returns a vector of indices that rearranges the given vector in ascending or descending order.

Vector can be used to subset a sorted dataframe.

Sorting in R

For Example, the below code sorts the data in ascending order of MSRP.



```
> # sorting in ascending order of MSRP
> columns <- c("Make", "Type","MSRP", "Invoice", "Horsepower","EngineSize")
> data[order(data$MSRP),columns]
 Make Type MSRP Invoice Horsepower EngineSize
413 Kia Sedan 10280 9875 104 1.6
63 Hyundai Sedan 10539 10107 103 1.6
154 Toyota Sedan 10760 10144 108 1.5
49 Saturn Sedan 10995 10319 140 2.2
167 Kia Sedan 11155 10705 104 1.6
125 Toyota Sedan 11290 10642 108 1.5
91 Toyota Sedan 11560 10896 108 1.5
79 Chevrolet Sedan 11690 10965 103 1.6
 [ reached 'max' / getOption("max.print") -- omitted 420 rows ]
```

Sorting Based on Multiple Columns

To sort data for multiple columns, each column vector shall be passed as a positional argument. It will result in a sorted data in the same order as the variable arguments.

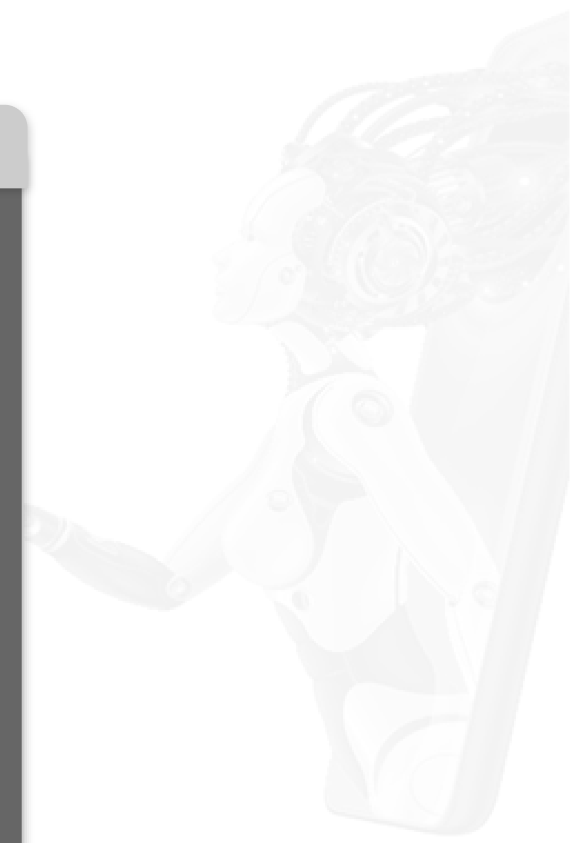
```
> # sorting first Make and MSRP within each Make
> # both descending
> columns <- c("Make", "Type","MSRP", "Invoice", "Horsepower","EngineSize")
> data[order(data$Make, data$MSRP, decreasing = TRUE),columns]
 Make Type MSRP Invoice Horsepower EngineSize
101 Volvo Sedan 45210 42573 268 2.9
260 Volvo Sedan 42565 40083 242 2.3
278 Volvo SUV 41250 38851 268 2.9
74 Volvo Sedan 40565 38203 197 2.4
124 Volvo Sedan 37885 35688 194 2.5
379 Volvo Sedan 37730 35542 208 2.9
217 Volvo Sedan 37560 35382 300 2.5
415 Volvo Wagon 35145 33112 208 2.5
 [ reached 'max' / getOption("max.print") -- omitted 420 rows ]
```

Data Summarizing

Descriptive Statistics

Statistical summary of data can be done using the summary() function.

```
> # let's get summary statistics for subset of the data including few factor and numerical data
> subset <- data[,c('Make', 'Type', 'Origin', 'MSRP', 'Length', 'Weight')]
> summary(subset)
 Make Type Origin MSRP Length Weight
 Toyota : 28 Hybrid: 3 : 15 Min. : 10280 Min. :143.0 Min. :1850
 Chevrolet : 27 Sedan :262 Asia :151 1st Qu.: 20354 1st Qu.:178.0 1st Qu.:3104
 Mercedes-Benz: 26 Sports: 49 Europe:119 Median : 27710 Median :187.0 Median :3474
 Ford : 23 SUV : 60 USA :143 Mean : 32789 Mean :186.4 Mean :3578
 BMW : 20 Truck : 24 3rd Qu.: 39350 3rd Qu.:194.0 3rd Qu.:3978
 Audi : 19 Wagon : 30 Max. :192465 Max. :238.0 Max. :7190
 (Other) :285 NA's :25
>
```



Important Statistical Functions for Data Analysis

Function	Description
mean(x)	Returns mean for a column of a dataframe
median(x)	Returns median for a column of a dataframe
sd(x)	Returns standard deviation for a column of a dataframe
var(x)	Returns variance for a column of a dataframe
table(x)	Returns count for each unique value in a data column
quantile(x)	Returns the 0th, 25th, 50th, 75th, and 100th percentiles of the data column
IQR(x)	Returns inter quartile range for a data column
range(x)	Returns minimum and maximum values for the data column

All these functions take vector (a column) as input.

Aggregation and Summarize

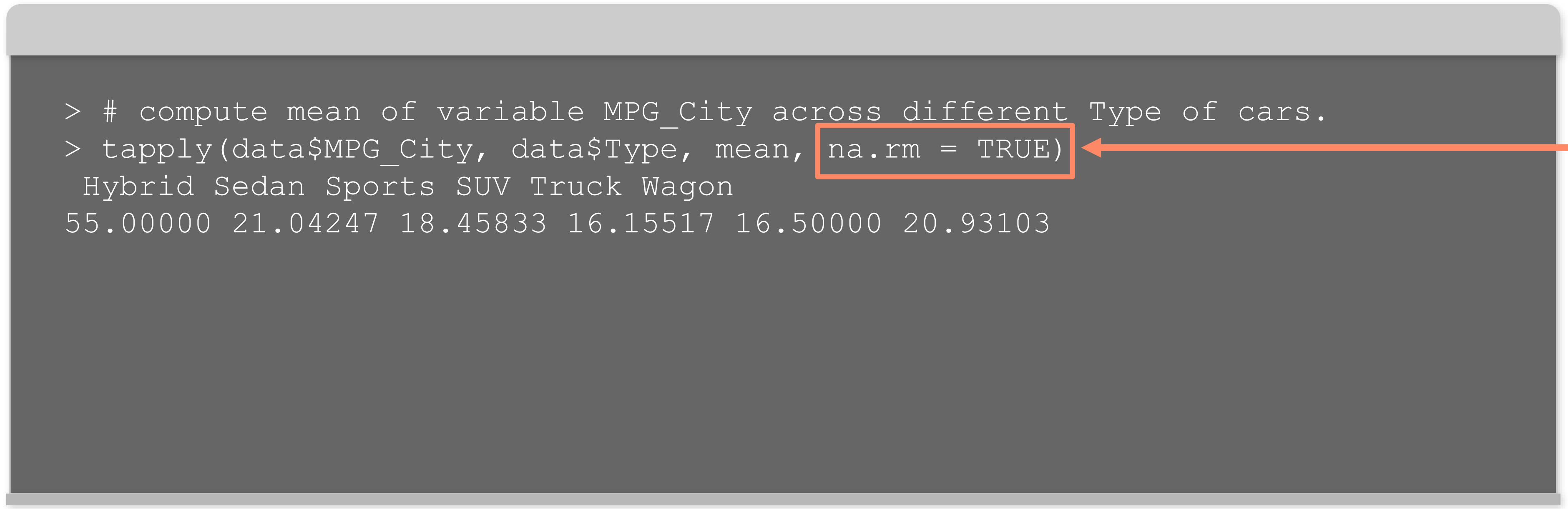
Aggregation of data provides more in-depth and effective data summarization. This process uses smaller groups of data, based on levels of a factor column, to compute summary statistics.

In R, aggregation can be done by using:

- aggregate()
- tapply()

tapply() for Aggregated Summary

tapply() summarizes only one data column across levels of only one-factor column.

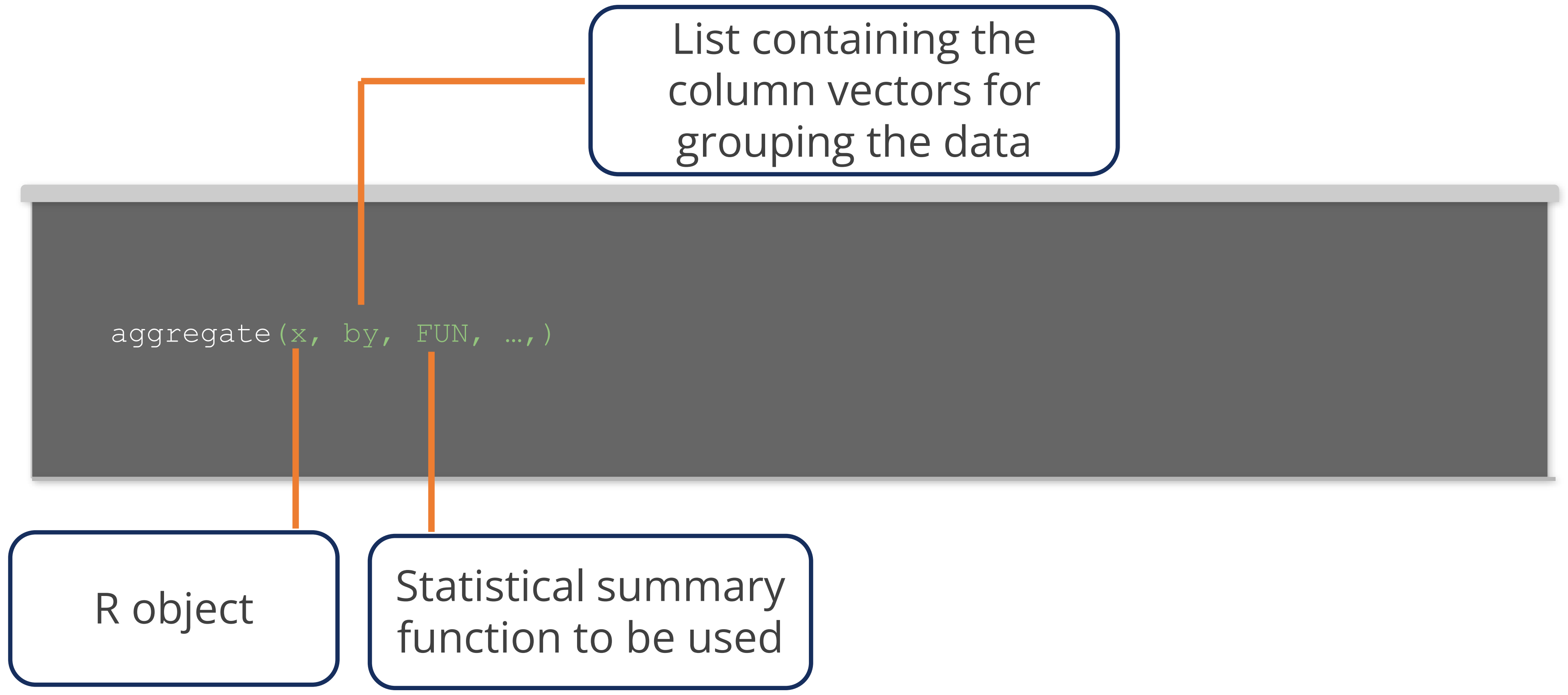


```
> # compute mean of variable MPG_City across different Type of cars.
> tapply(data$MPG_City, data$Type, mean, na.rm = TRUE)
 Hybrid Sedan Sports SUV Truck Wagon
55.00000 21.04247 18.45833 16.15517 16.50000 20.93103
```

Additional arguments to the applied functions can also be passed.

aggregate() for Aggregated Summary

The aggregate() function can aggregate multiple columns across combination of multiple factor columns.



```
aggregate(x, by, FUN, …,)
```

List containing the column vectors for grouping the data
R object
Statistical summary function to be used

aggregate() for Aggregated Summary

The example here computes the mean for variables MSRP and MPG_City across the types of cars from the “cars.csv” file.

```
> # compute mean for MSRP, MPG_City across Type.
> aggregate(x = data[c("MSRP", "MPG_City")],
+ by = list(data$Type),
+ FUN = mean, na.rm = TRUE)
 Group.1 MSRP MPG_City
1 Hybrid 20325.00 55.00000
2 Sedan 29715.79 21.04247
3 Sports 53793.15 18.45833
4 SUV 34447.41 16.15517
5 Truck 22967.18 16.50000
6 Wagon 29188.25 20.93103
```

Merging Data Tables

Merging of Data Tables

Merging is the process of combining data from two or more sources for analysis. Data for analysis might not be available as a single source.

“merge()” function in R combines two data tables at a time.

Syntax

```
merge(x,y, by = intersect(names(x), names(y)), all = FALSE)
```

Data is merged on the primary key, which is the set of data columns that are common in dataframes.

Merge

Following are the data sets for merging:

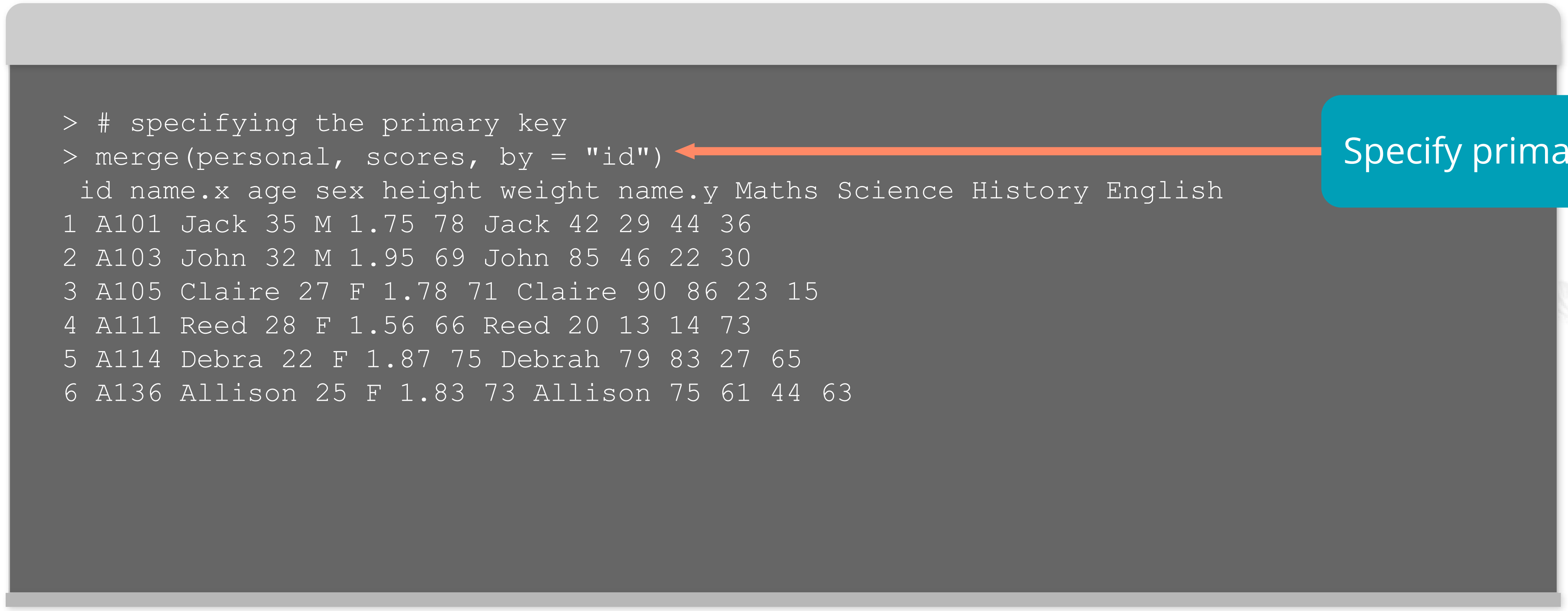
Personal: Consists of personal information as name, age, sex, height, weight

Scores: Contains scores in subjects: Maths, Science, History, and English with id and name as the common columns

```
> personal <- read.csv("personal.csv")
> scores <- read.csv("scores.csv")
> # merge detects common columns by self
> merge(personal, scores)
 id name age sex height weight Maths Science History English
1 A101 Jack 35 M 1.75 78 42 29 44 36
2 A103 John 32 M 1.95 69 85 46 22 30
3 A105 Claire 27 F 1.78 71 90 86 23 15
4 A111 Reed 28 F 1.56 66 20 13 14 73
5 A136 Allison 25 F 1.83 73 75 61 44 63
```

Merge

This code creates a dataframe by merging data from personal and scores files, matching them using their ids.



```
> # specifying the primary key
> merge(personal, scores, by = "id")
 id name.x age sex height weight name.y Maths Science History English
1 A101 Jack 35 M 1.75 78 Jack 42 29 44 36
2 A103 John 32 M 1.95 69 John 85 46 22 30
3 A105 Claire 27 F 1.78 71 Claire 90 86 23 15
4 A111 Reed 28 F 1.56 66 Reed 20 13 14 73
5 A114 Debra 22 F 1.87 75 Debrah 79 83 27 65
6 A136 Allison 25 F 1.83 73 Allison 75 61 44 63
```

Specify primary key as “id”

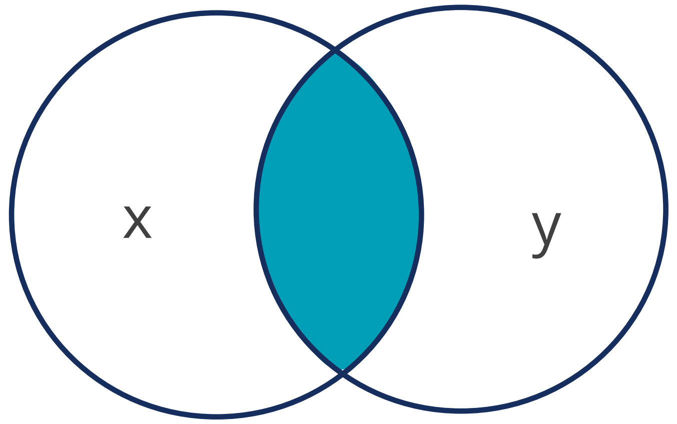
Types of Merge

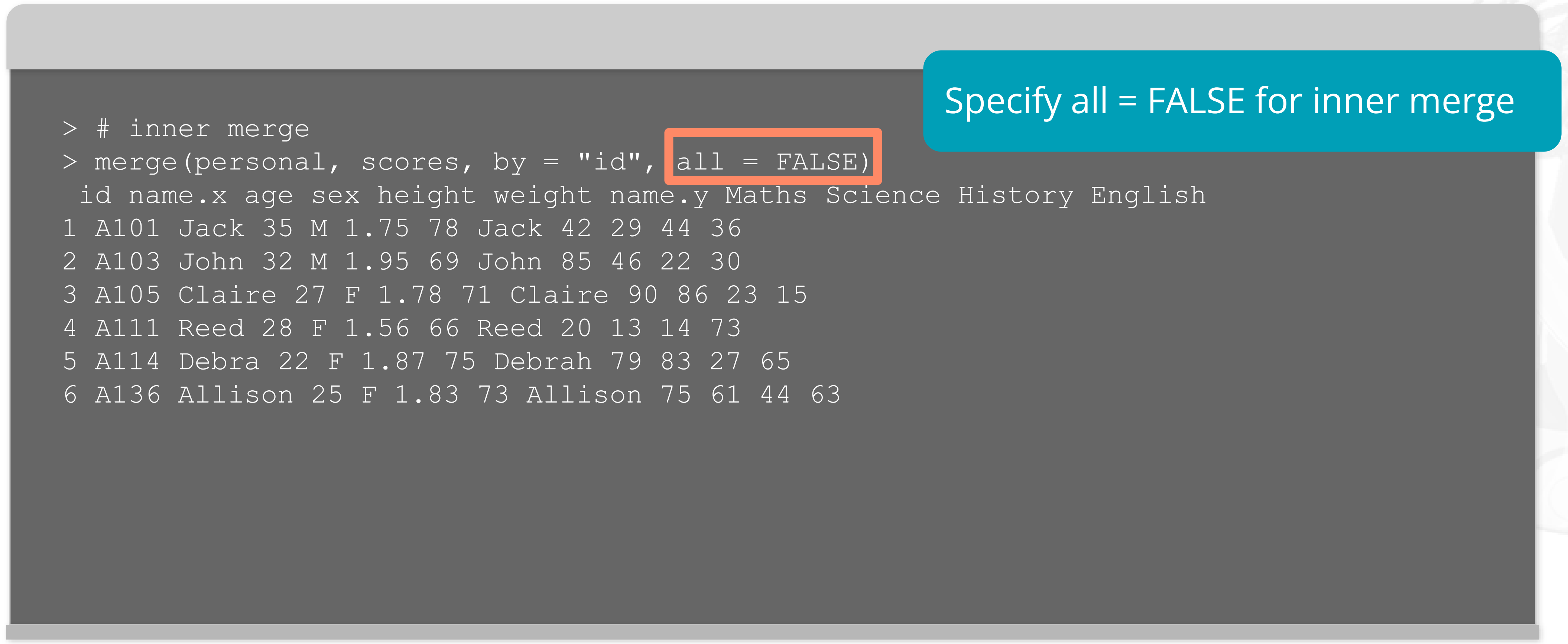
There are four types of merge:

- Inner merge
- Outer merge
- Left merge
- Right merge

Inner Merge

It combines dataframes to keep only the rows that match the primary key.



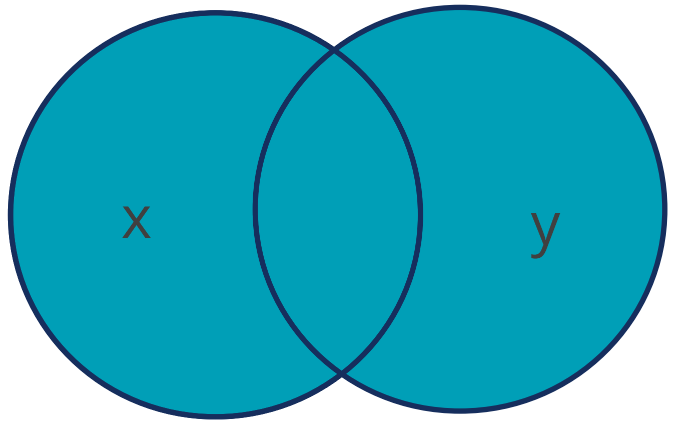


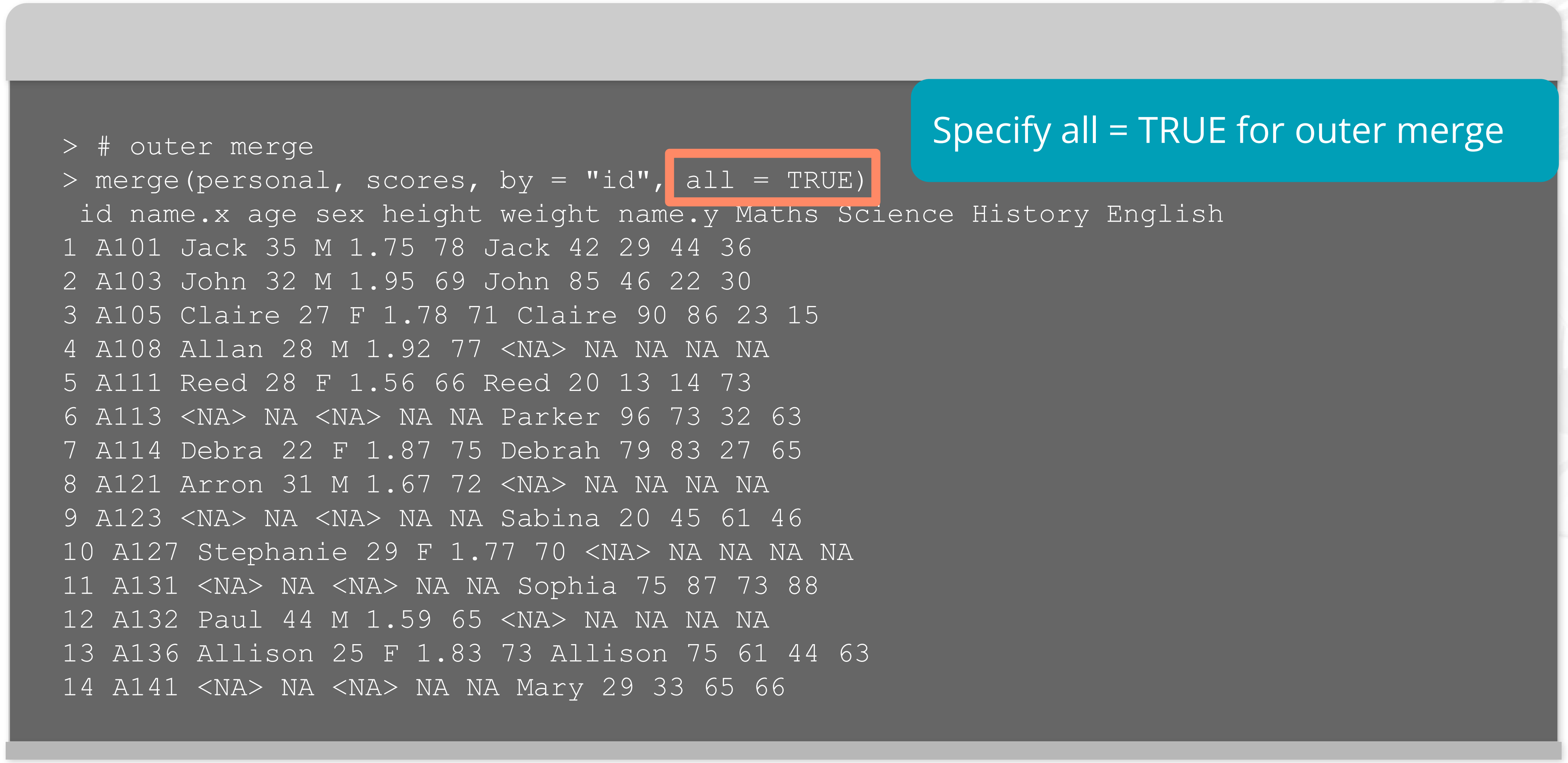
Specify all = FALSE for inner merge

```
> # inner merge
> merge(personal, scores, by = "id", all = FALSE)
 id name.x age sex height weight name.y Maths Science History English
1 A101 Jack 35 M 1.75 78 Jack 42 29 44 36
2 A103 John 32 M 1.95 69 John 85 46 22 30
3 A105 Claire 27 F 1.78 71 Claire 90 86 23 15
4 A111 Reed 28 F 1.56 66 Reed 20 13 14 73
5 A114 Debra 22 F 1.87 75 Debrah 79 83 27 65
6 A136 Allison 25 F 1.83 73 Allison 75 61 44 63
```

Outer Merge

It combines dataframes to keep all the rows from both the dataframes.



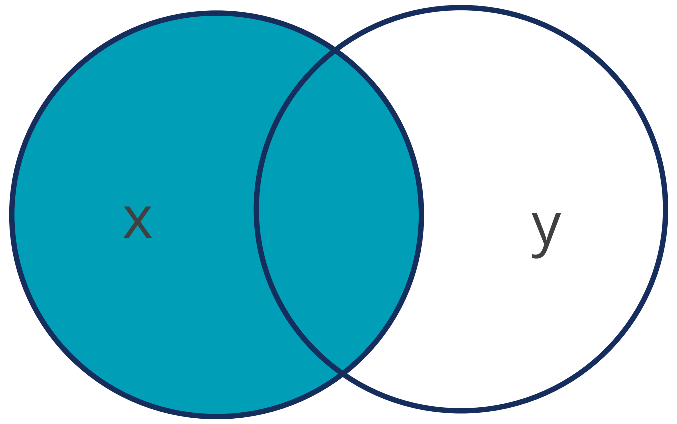


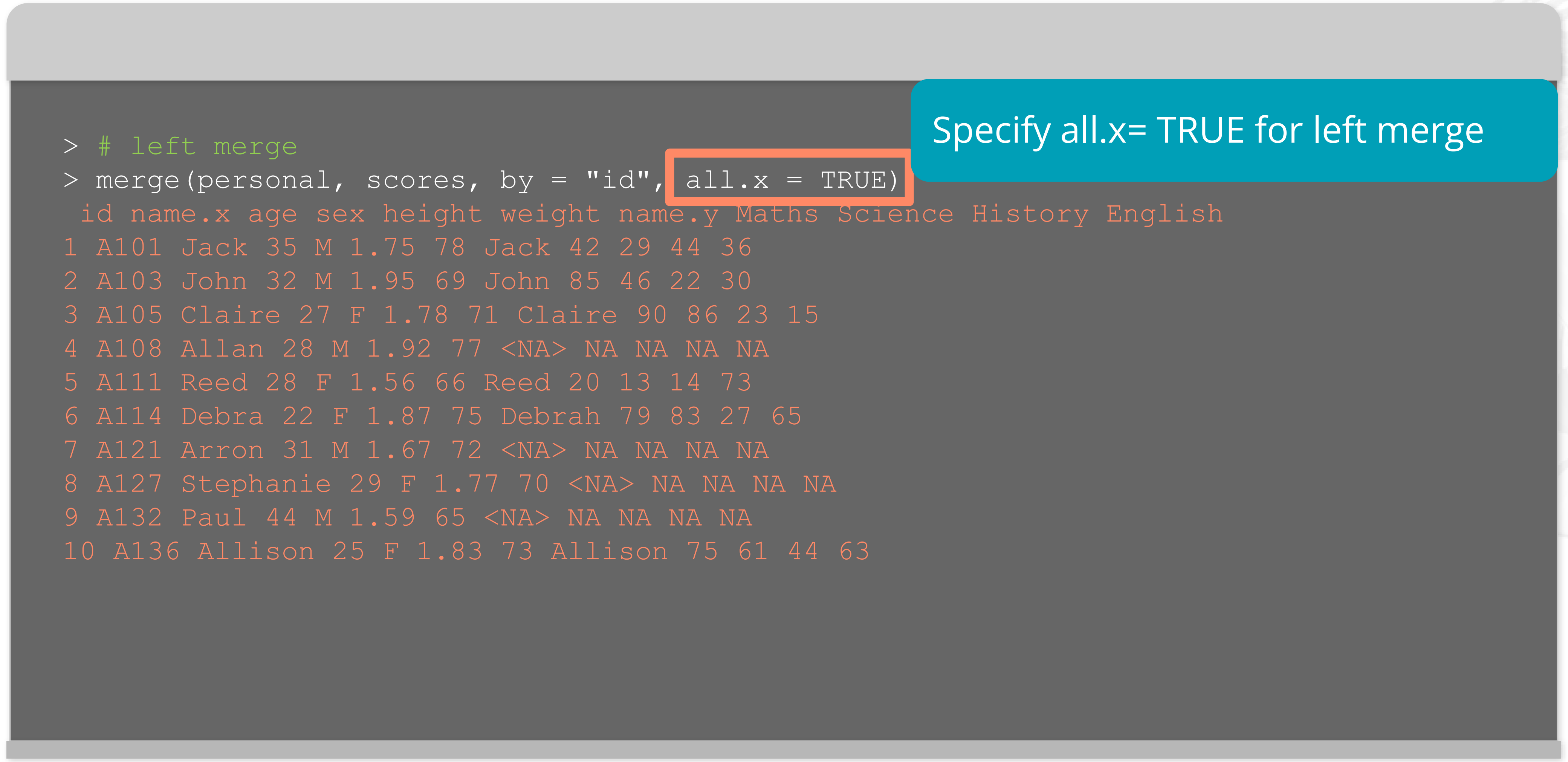
Specify all = TRUE for outer merge

```
> # outer merge
> merge(personal, scores, by = "id", all = TRUE)
 id name.x age sex height weight name.y Maths Science History English
1 A101 Jack 35 M 1.75 78 Jack 42 29 44 36
2 A103 John 32 M 1.95 69 John 85 46 22 30
3 A105 Claire 27 F 1.78 71 Claire 90 86 23 15
4 A108 Allan 28 M 1.92 77 <NA> NA NA NA NA
5 A111 Reed 28 F 1.56 66 Reed 20 13 14 73
6 A113 <NA> NA <NA> NA NA Parker 96 73 32 63
7 A114 Debra 22 F 1.87 75 Debrah 79 83 27 65
8 A121 Arron 31 M 1.67 72 <NA> NA NA NA NA
9 A123 <NA> NA <NA> NA NA Sabina 20 45 61 46
10 A127 Stephanie 29 F 1.77 70 <NA> NA NA NA NA
11 A131 <NA> NA <NA> NA NA Sophia 75 87 73 88
12 A132 Paul 44 M 1.59 65 <NA> NA NA NA NA
13 A136 Allison 25 F 1.83 73 Allison 75 61 44 63
14 A141 <NA> NA <NA> NA NA Mary 29 33 65 66
```

Left Merge

It combines dataframes to include all the rows of the dataframe ”x” and only the matching rows from the ”y” dataframe.



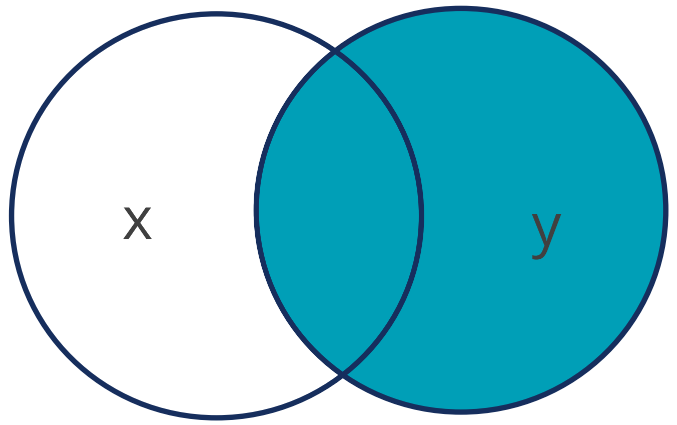


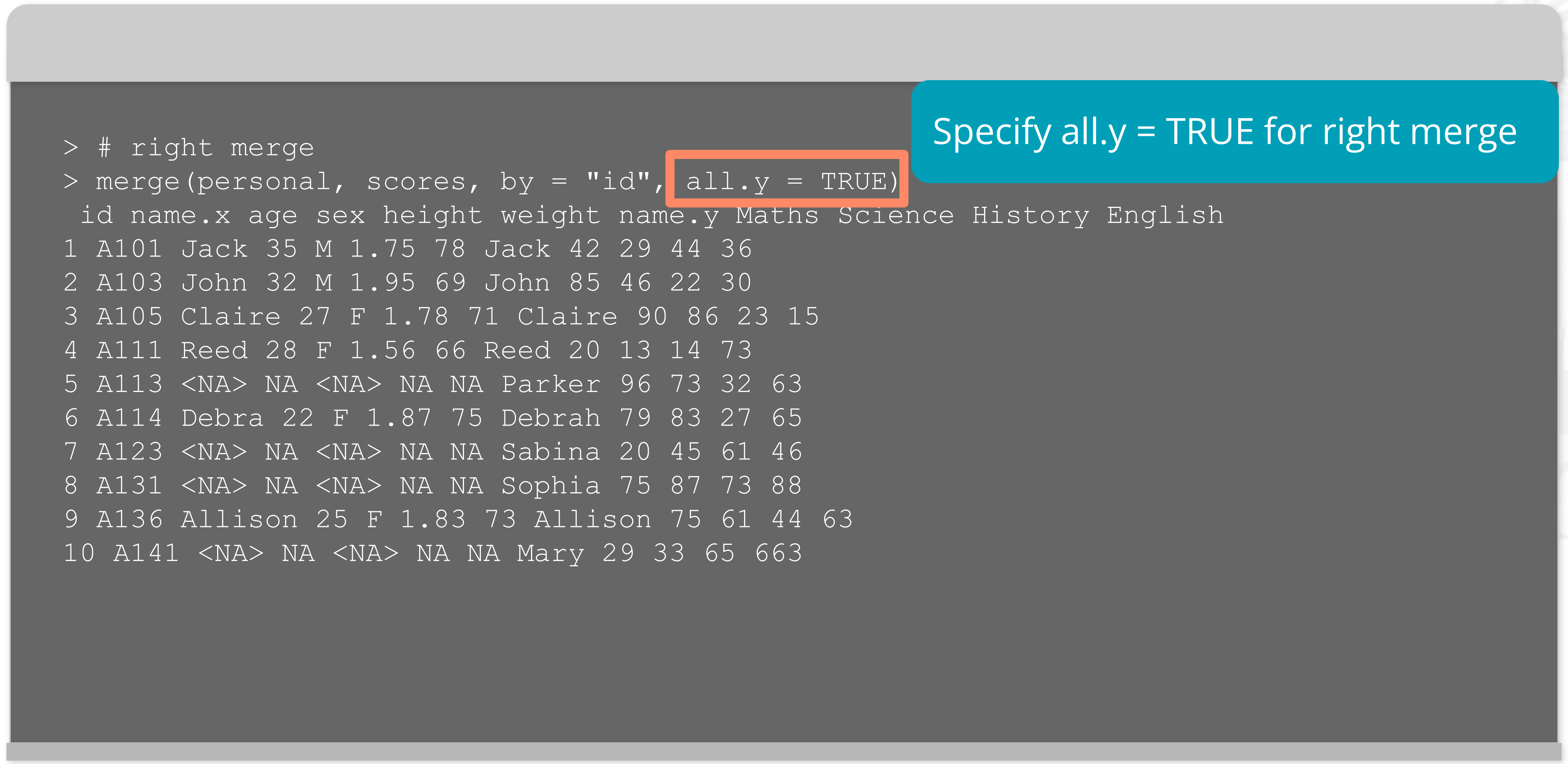
```
> # left merge
> merge(personal, scores, by = "id", all.x = TRUE)
 id name.x age sex height weight name.y Maths Science History English
1 A101 Jack 35 M 1.75 78 Jack 42 29 44 36
2 A103 John 32 M 1.95 69 John 85 46 22 30
3 A105 Claire 27 F 1.78 71 Claire 90 86 23 15
4 A108 Allan 28 M 1.92 77 <NA> NA NA NA NA
5 A111 Reed 28 F 1.56 66 Reed 20 13 14 73
6 A114 Debra 22 F 1.87 75 Debrah 79 83 27 65
7 A121 Arron 31 M 1.67 72 <NA> NA NA NA NA
8 A127 Stephanie 29 F 1.77 70 <NA> NA NA NA NA
9 A132 Paul 44 M 1.59 65 <NA> NA NA NA NA
10 A136 Allison 25 F 1.83 73 Allison 75 61 44 63
```

Specify all.x= TRUE for left merge

Right Merge

It combines dataframes to include all the rows of the dataframe ”y” and only the matching rows from ”x” dataframe.





```
> # right merge
> merge(personal, scores, by = "id", all.y = TRUE)
 id name.x age sex height weight name.y Maths Science History English
1 A101 Jack 35 M 1.75 78 Jack 42 29 44 36
2 A103 John 32 M 1.95 69 John 85 46 22 30
3 A105 Claire 27 F 1.78 71 Claire 90 86 23 15
4 A111 Reed 28 F 1.56 66 Reed 20 13 14 73
5 A113 <NA> NA <NA> NA NA Parker 96 73 32 63
6 A114 Debra 22 F 1.87 75 Debrah 79 83 27 65
7 A123 <NA> NA <NA> NA NA Sabina 20 45 61 46
8 A131 <NA> NA <NA> NA NA Sophia 75 87 73 88
9 A136 Allison 25 F 1.83 73 Allison 75 61 44 63
10 A141 <NA> NA <NA> NA NA Mary 29 33 65 663
```

Merging Data

Duration: 5 minutes

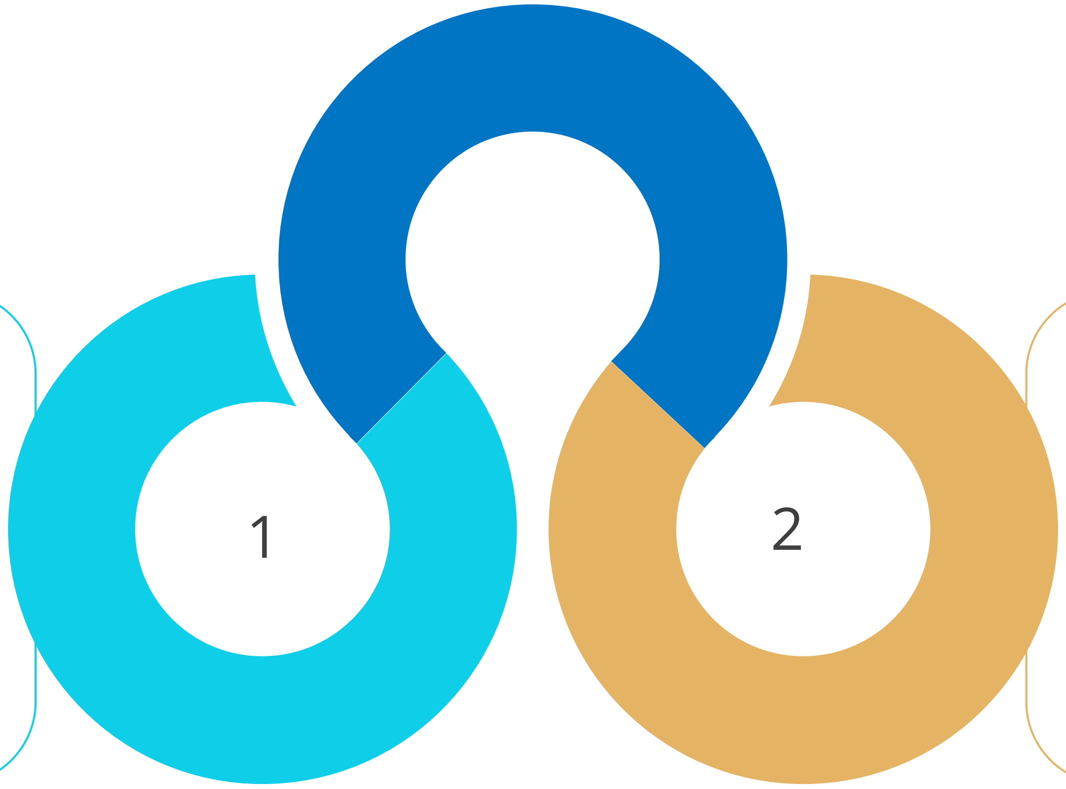
Problem Statement: Create a combined file from the ***Class_pers.csv*** and ***Class_marks.csv*** files based on the Enrolment Id and include only the common rows from the two datasets.

Note: Please download the data sets and the solution document from the Course Resources section and follow the steps given in the document

dplyr Package for Data Manipulation

The dplyr Package

It is described as the “grammar of data manipulation”.



It is a part of “tidyverse” packages.

dplyr functions provide verbs to perform data manipulation.

The dplyr Package

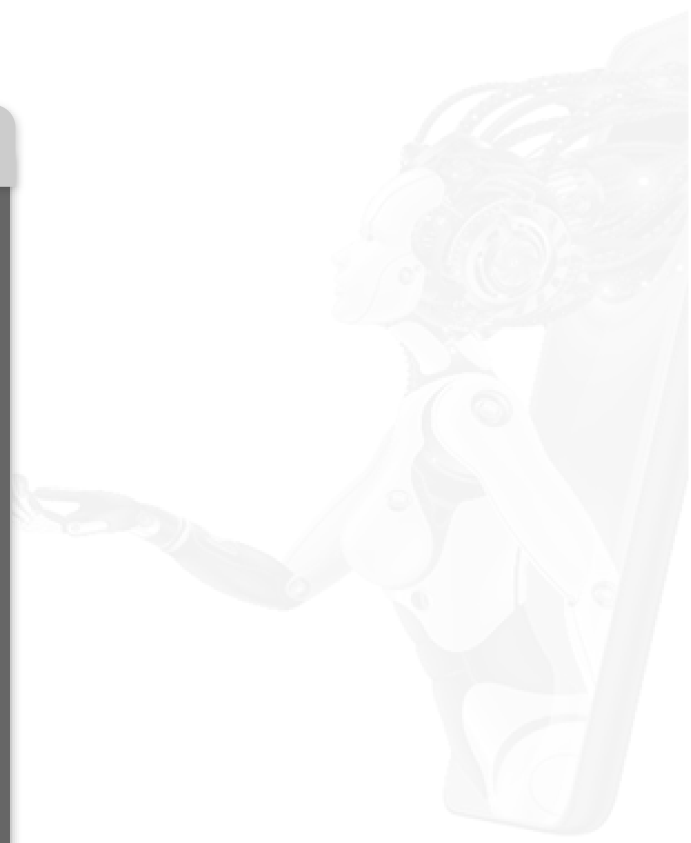
The most popular functions present in the dplyr package are:

- select
- filter
- mutate
- arrange
- groupby and summarise

The select() Function

The select() function is used to create a subset of data by selecting specific columns.

```
> # select specific columns
> select(data, Make, Type, MSRP, Horsepower, MPG_City)
 Make Type MSRP Horsepower MPG_City
1 Acura SUV 36945 265 17
2 Acura Sedan 23820 200 24
3 Acura Sedan 26990 200 22
4 Acura Sedan 33195 270 20
5 Acura Sedan 43755 225 18
 [ reached 'max' / getOption("max.print") -- omitted 423 rows ]
```



The select() Function

The dplyr package provides functions to be used with select function, to subset data for variables matching a pattern.

Function	Description
starts_with(x)	Selects variables if they start with given pattern x
ends_with(x)	Selects variables if they end with given pattern x
contains(x)	Selects variables if they contain given pattern x
match(expr)	Selects variables if they match a regular expression
num_range(prefix, range)	Matches a numerical range, such as, V1, V2, V3

The select() Function

The example here uses startswith() to select the variable names starting with “MPG” from the ***cars.csv*** file.

```
> # select specific columns starting with : MPG
> select(data, starts_with('MPG'))
 MPG_City MPG_Highway
1 17 23
2 24 31
3 22 29
4 20 28
5 18 24
6 18 24
7 17 24
 [ reached 'max' / getOption("max.print") -- omitted 421 rows ]
```

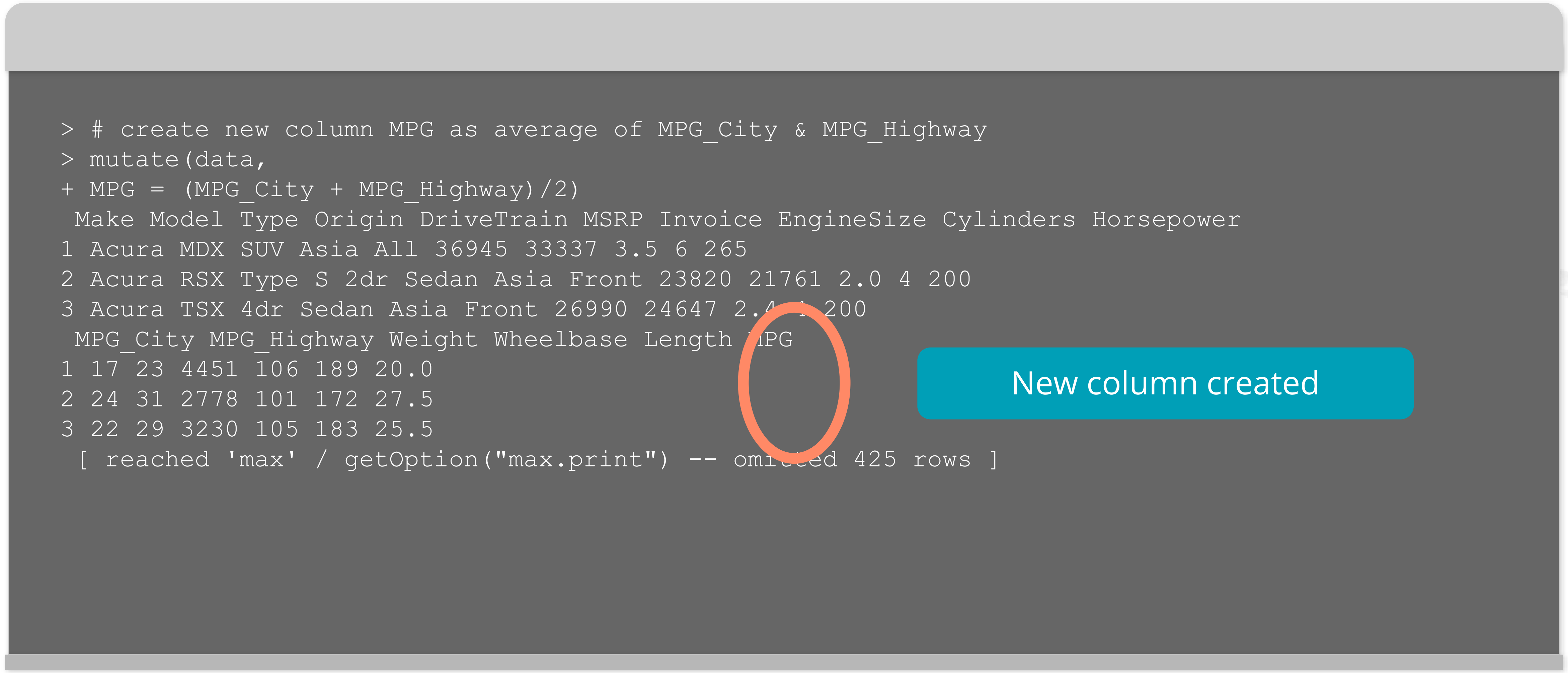
The filter() Function

The filter() function is used to create subset of data by conditional filtering.

```
> #single condition
> filter(data, MSRP > 100000)
 Make Model Type Origin DriveTrain MSRP Invoice EngineSize Cylinders
1 Mercedes-Benz CL600 2dr Sedan Europe Rear 128420 119600 5.5 12
2 Mercedes-Benz SL55 AMG 2dr Sports Europe Rear 121770 113388 5.5 8
3 Mercedes-Benz SL600 convertible 2dr Sports Europe Rear 126670 117854 5.5 12
4 Porsche 911 GT2 2dr Sports Europe Rear 192465 173560 3.6 6
 Horsepower MPG_City MPG_Highway Weight Wheelbase Length
1 493 13 19 4473 114 196
2 493 14 21 4235 101 179
3 493 13 19 4429 101 179
4 477 17 24 3131 93 175
```

The mutate() Function

The mutate() function is used to create new variables or modify existing ones. This function does not modify the data but returns a new dataframe.



```
> # create new column MPG as average of MPG_City & MPG_Highway
> mutate(data,
+ MPG = (MPG_City + MPG_Highway)/2)
 Make Model Type Origin DriveTrain MSRP Invoice EngineSize Cylinders Horsepower
1 Acura MDX SUV Asia All 36945 33337 3.5 6 265
2 Acura RSX Type S 2dr Sedan Asia Front 23820 21761 2.0 4 200
3 Acura TSX 4dr Sedan Asia Front 26990 24647 2.4 4 200
 MPG_City MPG_Highway Weight Wheelbase Length MPG
1 17 23 4451 106 189 20.0
2 24 31 2778 101 172 27.5
3 22 29 3230 105 183 25.5
 [ reached 'max' / getOption("max.print") -- omitted 425 rows ]
```

The arrange() Function

It creates a sorted data based on the given variables. To sort in descending order, variable shall be given as desc(var_name).

```
> # sort data for Make and then descending order of MSRP
> arrange(data, Make, desc(MSRP))
 Make Model Type Origin DriveTrain MSRP Invoice EngineSize Cylinders
1 Acura NSX coupe 2dr manual S Sports Asia Rear 89765 79978 3.2 6
2 Acura 3.5 RL w/Navigation 4dr Sedan Asia Front 46100 41100 3.5 6
3 Acura 3.5 RL 4dr Sedan Asia Front 43755 39014 3.5 6
4 Acura MDX SUV Asia All 36945 33337 3.5 6
5 Acura TL 4dr Sedan Asia Front 33195 30299 3.2 6
6 Acura TSX 4dr Sedan Asia Front 26990 24647 2.4 4
 Horsepower MPG_City MPG_Highway Weight Wheelbase Length
1 290 17 24 3153 100 174
2 225 18 24 3893 115 197
3 225 18 24 3880 115 197
4 265 17 23 4451 106 189
5 270 20 28 3575 108 186
6 200 22 29 3230 105 183
 [ reached 'max' / getOption("max.print") -- omitted 422 rows ]
```

summarise() and group_by()

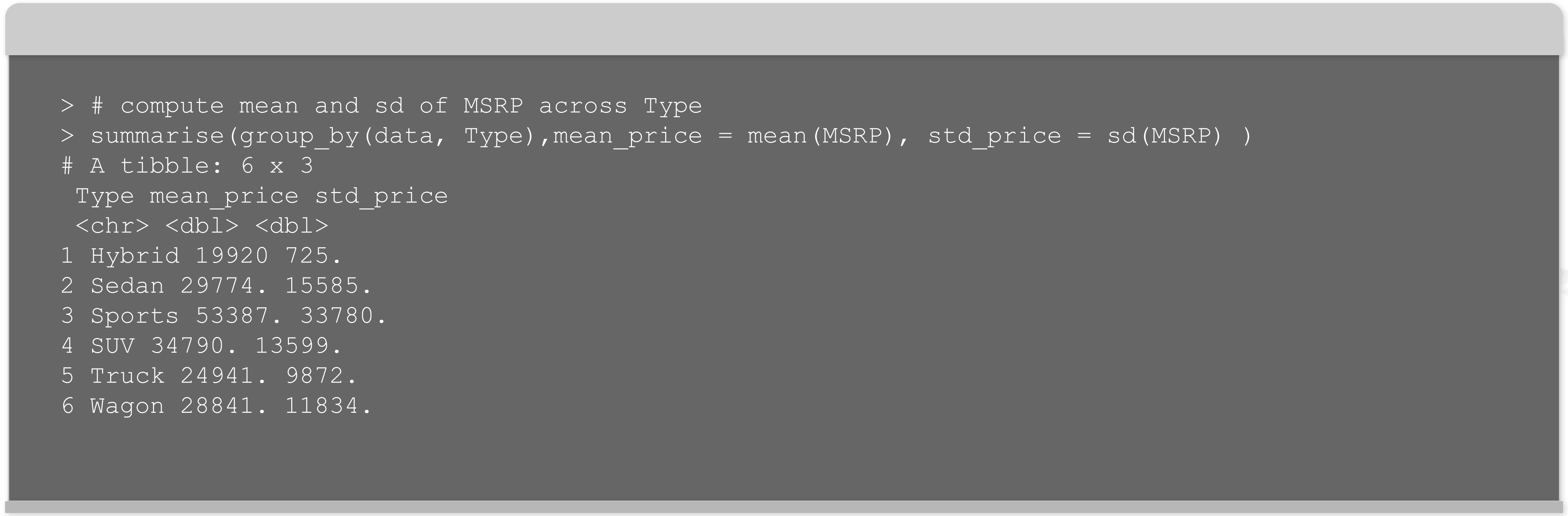
The summarise() function provides summary statistic for the data using summary functions such as mean(), median(), sd().

When combined with group_by(), it yields meaningful aggregated results for each categorical group.

These functions provide much more functionality than the tradition aggregation functions.

summarise() and group_by()

Example: The function here computes mean and standard deviation for given variable(s).



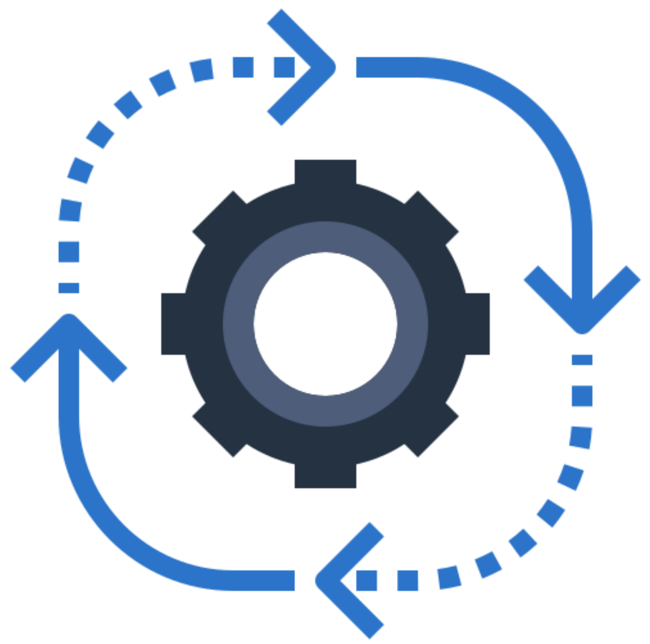
```
> # compute mean and sd of MSRP across Type
> summarise(group_by(data, Type),mean_price = mean(MSRP), std_price = sd(MSRP) )
# A tibble: 6 x 3
 Type mean_price std_price
 <chr> <dbl> <dbl>
1 Hybrid 19920 725.
2 Sedan 29774. 15585.
3 Sports 53387. 33780.
4 SUV 34790. 13599.
5 Truck 24941. 9872.
6 Wagon 28841. 11834.
```

Scoped Verbs

Each of the verb in dplyr comes in three additional forms suffixed with _if, _at, and _all.

Scoped verb	Description	Sample functions
_if	Picks variables based on functions like is.numeric(), is.character()	select_if(), mutate_if(), summarise_if()
_at	Picks specific variables as in select() function	select_at(), summarise_at() mutate_at()
_all	Performs operation on all the variables	select_all, summarise_all(), mutate_all()

Pipeline Operator : %>%



dplyr provides special operator to create pipeline of tasks to be performed.

The output of the first expression acts as the first positional argument to the subsequent expression.

Pipeline operator can be used to link functions from any package.

Pipeline

```
> # combining filter and selct opertions
> data %>%
+ filter(MSRP > 100000) %>%
+ select(Make, Type, MSRP, Invoice, MPG_City, MPG_Highway) %>%
+ arrange(desc(MSRP))
 Make Type MSRP Invoice MPG_City MPG_Highway
1 Porsche Sports 192465 173560 17 24
2 Mercedes-Benz Sedan 128420 119600 13 19
3 Mercedes-Benz Sports 126670 117854 13 19
4 Mercedes-Benz Sports 121770 113388 14 21
```

Data is first argument to filter function.

↓

The filtered data is now the input to the select function.

↓

The selected data is now arranged in descending order of MSRP.

Key Takeaways

- Data wrangling is a process of gathering, transforming, and preparing data for further detailed analysis.
- R can import data from various formats such as CSV, Excel, SAS, SPSS, etc.
- dplyr is the grammar of data manipulation. It helps perform basic data manipulation functions with ease.
- Pipeline operators from the dplyr package help create pipeline of tasks to be performed.

Knowledge Check

Knowledge Check

1

Which of the statements is true?

A. Data wrangling is a process designed to transform raw data into a more useable format.

B. Data wrangling involves gathering, restructuring, and cleaning the data.

C. Both A and B

D. None of the above

Knowledge Check

1

Which of the statements is true?

A. Data wrangling is a process designed to transform raw data into a more useable format.

B. Data wrangling involves gathering, restructuring, and cleaning the data.

C. Both A and B

D. None of the above

The correct answer is C

Data wrangling is a process designed to transform raw data into more useable format that involves gathering, restructuring, and cleaning the data.

Knowledge Check

2

Consider the below code:

merge(x, y, by = ‘id’, all.x = TRUE)

What does this function do?

A. Inner merge

B. Outer merge

C. Left merge

D. Right merge

Knowledge Check

2

Consider the below code:

```
merge(x, y, by = ‘id’, all.x = TRUE)
```

What does this function do?

A. Inner merge

B. Outer merge

C. Left merge

D. Right merge

The correct answer is C

all.x = TRUE performs left merge as all the data observations from left data, that is, “x” is included.

Knowledge Check

3

Which of the functions will summarize all the numeric data variables?

A. summarise_if(mtcars, is.numeric, mean, na.rm = TRUE)

B. summarise_at(mtcars, is.numeric, mean, na.rm = TRUE)

C. summarise_all(mtcars, is.numeric, mean, na.rm = TRUE)

D. None of the above

Knowledge Check

3

Which of the functions will summarize all the numeric data variables?

A. summarise_if(mtcars, is.numeric, mean, na.rm = TRUE)

B. summarise_at(mtcars, is.numeric, mean, na.rm = TRUE)

C. summarise_all(mtcars, is.numeric, mean, na.rm = TRUE)

D. None of the above

The correct answer is A

The _if suffix with summarise() function selects variables from given data based on the predicate function and summarizes the data.