**Get Date and Time in different Formats in R Programming – date(), Sys.Date(), Sys.time() and Sys.timezone() Function**

**date()** function in [R Language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) is used to return the current date and time.

***Syntax:****date()*

***Parameters:*** *Does not accept any parameters*

**Example:**

|  |
| --- |
| # R program to illustrate  # date function    # Calling date() function to  # return current date and time  date() |

**Output:**

[1] "Thu Jun 11 04:29:39 2020"

**Sys.Date() Function**

**Sys.Date()** function is used to return the system’s date.

***Syntax:****Sys.Date()*

***Parameters:*** *Does not accept any parameters*

**Example:**

|  |
| --- |
| # R program to illustrate  # Sys.Date function    # Calling Sys.Date() function to  # return the system's date  Sys.Date() |

**Output:**

[1] "2020-06-11"

**Sys.time()**

**Sys.time()** function is used to return the system’s date and time.

***Syntax:****Sys.time()*

***Parameters:*** *Does not accept any parameters*

**Example:**

|  |
| --- |
| # R program to illustrate  # Sys.time function    # Calling Sys.time() function to  # return the system's date and time  Sys.time() |

**Output:**

[1] "2020-06-11 05:35:49 UTC"

**Sys.timezone()**

**Sys.timezone()** function is used to return the current time zone.

***Syntax:****Sys.timezone()*

***Parameters:*** *Does not accept any parameters*

**Example:**

|  |
| --- |
| # R program to illustrate  # Sys.timezone function    # Calling Sys.timezone() function to  # return the current time zone  Sys.timezone() |

**Output:**

[1] "Etc/UTC"

# dplyr Package in R Programming

In this article, we will discuss Aggregating and analyzing data with dplyr package in the R Programming Language.

## dplyr Package in R

**The dplyr package in**[**R Programming Language**](https://www.geeksforgeeks.org/introduction-to-r-programming-language/)is a structure of data manipulation that provides a uniform set of verbs, helping to resolve the most frequent data manipulation hurdles.

* By limiting the choices the focus can now be more on data manipulation difficulties.
* There are uncomplicated “verbs”, functions present for tackling every common data manipulation and the thoughts can be translated into code faster.
* There are valuable backends and hence waiting time for the computer is reduced.

Here are some key functions and concepts within the dplyr package in R.

### **Data Frame and Tibble**

Data frames in dplyr in R is organized tables where each column stores specific types of information, like names, ages, or scores. for creating a data frame involves specifying column names and their respective values.

* R

|  |
| --- |
| df <- data.frame(    Name = c("vipul", "jayesh", "anurag"),    Age = c(25, 23, 22),    Score = c(95, 89, 78)  )  Df |

**Output:**

Name Age Score  
1 vipul 25 95  
2 jayesh 23 89  
3 anurag 22 78

On the other hand, tibbles, introduced through the [tibble package](https://www.geeksforgeeks.org/data-wrangling-in-r-programming-working-with-tibbles/" \t "_blank), share similar functionality but offer enhanced user-friendly features. The syntax for creating a tibble is comparable to that of a data frame.

### **Pipes (%>%)**

dplyr in R The pipe operator (%>%) in dplyr package, which allows us to chain multiple operations together, improving code readability.

* R

|  |
| --- |
| # Load necessary libraries  library(dplyr)    # Example: Chain operations using the pipe operator  result <- mtcars %>%    filter(mpg > 20) %>%        # Filter rows where mpg is greater than 20    select(mpg, cyl, hp) %>%    # Select specific columns    group\_by(cyl) %>%           # Group the data by the 'cyl' variable    summarise(mean\_hp = mean(hp))  # Calculate the mean horsepower for each group    # Display the result  print(result) |

**Output:**

cyl mean\_hp  
 <dbl> <dbl>  
1 4 82.6  
2 6 110

### Verb Functions

dplyr in R provides various important functions that can be used for Data Manipulation. These are:

### **filter() Function**

For choosing cases and using their values as a base for doing so.

* R

|  |
| --- |
| # Create a data frame with missing data  d <- data.frame(name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),                  age = c(7, 5, 9, 16),                  ht = c(46, **NA**, **NA**, 69),                  school = c("yes", "yes", "no", "no"))    # Display the data frame  print(d)    # Finding rows with NA value  rows\_with\_na <- d %>% filter(is.na(ht))  print(rows\_with\_na)    # Finding rows with no NA value  rows\_without\_na <- d %>% filter(!is.na(ht))  print(rows\_without\_na) |

**Output:**

name age ht school  
1 Abhi 7 46 yes  
2 Bhavesh 5 NA yes  
3 Chaman 9 NA no  
4 Dimri 16 69 no  
Finding rows with NA value  
 name age ht school  
1 Bhavesh 5 NA yes  
2 Chaman 9 NA no  
Finding rows with no NA value  
 name age ht school  
1 Abhi 7 46 yes  
2 Dimri 16 69 no

### **arrange():**

For reordering of the cases.

* R

|  |
| --- |
| # Create a data frame with missing data  d <- data.frame( name = c("Abhi", "Bhavesh", "Chaman", "Dimri"),                   age = c(7, 5, 9, 16),                   ht = c(46, **NA**, **NA**, 69),                   school = c("yes", "yes", "no", "no") )  d    # Arranging name according to the age  d.name<- arrange(d, age)  print(d.name) |

**Output:**

name age ht school  
1 Abhi 7 46 yes  
2 Bhavesh 5 NA yes  
3 Chaman 9 NA no  
4 Dimri 16 69 no  
   
Arranging name according to the age  
 name age ht school  
1 Bhavesh 5 NA yes  
2 Abhi 7 46 yes  
3 Chaman 9 NA no  
4 Dimri 16 69 no

### **select() and rename():**

For choosing variables and using their names as a base for doing so.

* R

|  |
| --- |
| # Create a data frame with missing data  d <- data.frame(name=c("Abhi", "Bhavesh",                          "Chaman", "Dimri"),                   age=c(7, 5, 9, 16),                   ht=c(46, **NA**, **NA**, 69),                   school=c("yes", "yes", "no", "no"))    # startswith() function to print only ht data  select(d, starts\_with("ht"))    # -startswith() function to print  # everything except ht data  select(d, -starts\_with("ht"))    # Printing column 1 to 2  select(d, 1: 2)    # Printing data of column  # heading containing 'a'  select(d, contains("a"))    # Printing data of column  # heading which matches 'na'  select(d, matches("na")) |

**Output:**

ht  
1 46  
2 NA  
3 NA  
4 69  
everything except ht data  
 name age school  
1 Abhi 7 yes  
2 Bhavesh 5 yes  
3 Chaman 9 no  
4 Dimri 16 no  
Printing column 1 to 2  
 name age  
1 Abhi 7  
2 Bhavesh 5  
3 Chaman 9  
4 Dimri 16  
heading containing 'a'  
 name age  
1 Abhi 7  
2 Bhavesh 5  
3 Chaman 9  
4 Dimri 16  
heading which matches 'na'  
 name  
1 Abhi  
2 Bhavesh  
3 Chaman  
4 Dimri

### **mutate() and transmute():**

Addition of new variables which are the functions of prevailing variables.

* R

|  |
| --- |
| # Create a data frame with missing data  d <- data.frame( name = c("Abhi", "Bhavesh",                            "Chaman", "Dimri"),                   age = c(7, 5, 9, 16),                   ht = c(46, **NA**, **NA**, 69),                   school = c("yes", "yes", "no", "no") )    # Calculating a variable x3 which is sum of height  # and age printing with ht and age  mutate(d, x3 = ht + age)    # Calculating a variable x3 which is sum of height  # and age printing without ht and age  transmute(d, x3 = ht + age) |

**Output:**

name age ht school  
1 Abhi 7 46 yes  
2 Bhavesh 5 NA yes  
3 Chaman 9 NA no  
4 Dimri 16 69 no  
Calculating a variable x3 which is sum of height  
   
 name age ht school x3  
1 Abhi 7 46 yes 53  
2 Bhavesh 5 NA yes NA  
3 Chaman 9 NA no NA  
4 Dimri 16 69 no 85  
Calculating a variable x3 which is sum of height   
 x3  
1 53  
2 NA  
3 NA  
4 85

### **summarise():**

Condensing various values to one value.

* R

|  |
| --- |
| # Create a data frame with missing data  d <- data.frame( name = c("Abhi", "Bhavesh",                            "Chaman", "Dimri"),                   age = c(7, 5, 9, 16),                   ht = c(46, **NA**, **NA**, 69),                   school = c("yes", "yes", "no", "no") )    # Calculating mean of age  summarise(d, mean = mean(age))    # Calculating min of age  summarise(d, med = min(age))    # Calculating max of age  summarise(d, med = max(age))    # Calculating median of age  summarise(d, med = median(age)) |

**Output:**

Calculating mean of age  
 mean  
1 9.25  
Calculating minimum age  
 med  
1 5  
Calculating max of age  
 med  
1 16  
Calculating median of age  
 med  
1 8

### **sample\_n() and sample\_frac():**

For taking random specimens.

* R

|  |
| --- |
| # Create a data frame with missing data  d <- data.frame( name = c("Abhi", "Bhavesh",                            "Chaman", "Dimri"),                   age = c(7, 5, 9, 16),                   ht = c(46, **NA**, **NA**, 69),                   school = c("yes", "yes", "no", "no") )    # Printing three rows  sample\_n(d, 3)    # Printing 50 % of the rows  sample\_frac(d, 0.50) |

**Output:**

name age ht school  
1 Chaman 9 NA no  
2 Dimri 16 69 no  
3 Abhi 7 46 yes  
 Printing 50 % of the rows  
 name age ht school  
1 Abhi 7 46 yes  
2 Dimri 16 69 no

# Tidyr Package in R Programming

[Packages](https://www.geeksforgeeks.org/packages-in-r-programming/) in the [R language](https://www.geeksforgeeks.org/introduction-to-r-programming-language/) are a collection of R functions, compiled code, and sample data. They are stored under a directory called **“library”** in the R environment. By default, R installs a set of packages during installation.  One of the most important packages in R is the **tidyr** package. The sole purpose of the **tidyr** package is to simplify the process of creating **tidy data**.

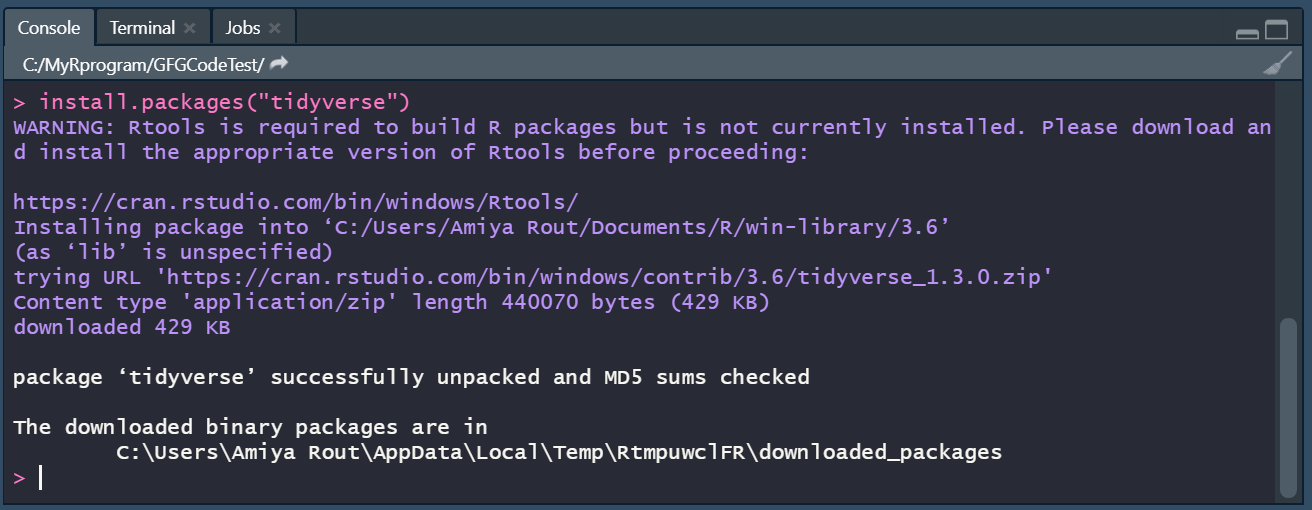
the ideal format in which to analyze data tables is called tidy data. A tabular dataset is tidy when each column corresponds to one variable in the dataset, each row corresponds to one observation, and all variables in the dataset have the same unit of observation.

Tidy data describes a standard way of storing data that is used wherever possible throughout the **tidyverse**. If you once make sure that your data is tidy, you’ll spend less time punching with the tools and more time working on your analysis.

### **Installation**

To use a package in R programming one must have to install the package first. This task can be done using the command **install.packages(“packagename”)**. To install the whole **tidyverse** package type this:

install.packages("tidyverse")



Alternatively, to install just **tidyr** package type this:

install.packages("tidyr")

To install the development version from GitHub type this:

# install.packages("devtools")

devtools::install\_github("tidyverse/tidyr")

### **Important Verb Functions in tidyr Package**

**The Dataset:**

Before going to the important verb function let’s prepare the data set first. Define a dataset **tidy\_dataframe** that contains data about the frequency of people in a particular group.

* R

|  |
| --- |
| # load the tidyr package  library(tidyr)    n = 10  # creating a data frame  tidy\_dataframe = data.frame(                        S.No = c(1:n),                      Group.1 = c(23, 345, 76, 212, 88,                                  199, 72, 35, 90, 265),                      Group.2 = c(117, 89, 66, 334, 90,                                 101, 178, 233, 45, 200),                      Group.3 = c(29, 101, 239, 289, 176,                                  320, 89, 109, 199, 56))    # print the elements of the data frame  tidy\_dataframe |

**Output:**

S.No Group.1 Group.2 Group.3

1 1 23 117 29

2 2 345 89 101

3 3 76 66 239

4 4 212 334 289

5 5 88 90 176

6 6 199 101 320

7 7 72 178 89

8 8 35 233 109

9 9 90 45 199

10 10 265 200 56

tidyr package provides various important functions that can be used for Data Cleaning. Those are:

* **gather() function:**It takes multiple columns and gathers them into key-value pairs. Basically it makes “wide” data longer. The **gather()** function will take multiple columns and collapse them into key-value pairs, duplicating all other columns as needed.

***Syntax:***

*gather(data, key = “key”, value = “value”, …, na.rm = FALSE, convert = FALSE, factor\_key = FALSE)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | the data frame. |
| key, value | the names of new key and value columns,  as strings or as symbols. |
| ……. | the selection of columns. If left empty, all variables are selected.  You can supply bare variable names, select all variables between   x and z with x:z, exclude y with -y. |
| na.rm | if set TRUE, it will remove rows from output where the value column is NA. |
| convert | is set TRUE, it will automatically run type.convert() on the key column.  This is useful if the column types are actually numeric,   integer, or logical. |
| factor\_key | if FALSE, the default, the key values will be stored as a character vector.   If TRUE, will be stored as a factor, which preserves  the original ordering of the columns. |

**Example:**

Now for a better understanding, we will make our data long with **gather()** function.

* R

|  |
| --- |
| # using gather() function on tidy\_dataframe  long <- tidy\_dataframe %>%              gather(Group, Frequency,                     Group.1:Group.3)    # print the data frame in a long format  long |

**Output:**

S.No Group Frequency

1 1 Group.1 23

2 2 Group.1 345

3 3 Group.1 76

4 4 Group.1 212

5 5 Group.1 88

6 6 Group.1 199

7 7 Group.1 72

8 8 Group.1 35

9 9 Group.1 90

10 10 Group.1 265

11 1 Group.2 117

12 2 Group.2 89

13 3 Group.2 66

14 4 Group.2 334

15 5 Group.2 90

16 6 Group.2 101

17 7 Group.2 178

18 8 Group.2 233

19 9 Group.2 45

20 10 Group.2 200

21 1 Group.3 29

22 2 Group.3 101

23 3 Group.3 239

24 4 Group.3 289

25 5 Group.3 176

26 6 Group.3 320

27 7 Group.3 89

28 8 Group.3 109

29 9 Group.3 199

30 10 Group.3 56

* **separate() function:**It converts longer data to a wider format. The **separate()** function turns a single character column into multiple columns.

***Syntax:***

*separate(data, col, into, sep = ” “, remove = TRUE, convert = FALSE)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| col | Column name or position. |
| into | Names of new variables to create as character vector.  Use NA to omit the variable in the output. |
| sep | The separator between the columns. |
| remove | If set TRUE, it will remove input column from the output data frame. |
| convert | If TRUE, will run type.convert() with as.is = TRUE on new columns. |

**Example:**

We can say that the long datasets created using **gather()** is appropriate for use, but we can break down Group variable even further using **separate()**.

* R

|  |
| --- |
| # import tidyr package  library(tidyr)  long <- tidy\_dataframe %>%              gather(Group, Frequency,                     Group.1:Group.3)    # use separate() function to make data wider  separate\_data <- long %>%              separate(Group, c("Allotment",                                "Number"))    # print the wider format  separate\_data |

**Output:**

S.No Allotment Number Frequency

1 1 Group 1 23

2 2 Group 1 345

3 3 Group 1 76

4 4 Group 1 212

5 5 Group 1 88

6 6 Group 1 199

7 7 Group 1 72

8 8 Group 1 35

9 9 Group 1 90

10 10 Group 1 265

11 1 Group 2 117

12 2 Group 2 89

13 3 Group 2 66

14 4 Group 2 334

15 5 Group 2 90

16 6 Group 2 101

17 7 Group 2 178

18 8 Group 2 233

19 9 Group 2 45

20 10 Group 2 200

21 1 Group 3 29

22 2 Group 3 101

23 3 Group 3 239

24 4 Group 3 289

25 5 Group 3 176

26 6 Group 3 320

27 7 Group 3 89

28 8 Group 3 109

29 9 Group 3 199

30 10 Group 3 56

* **unite() function:**It merges two columns into one column. The **unite()** function is a convenience function to paste together multiple variable values into one. In essence, it combines two variables of a single observation into one variable.

***Syntax:***

*unite(data, col, …, sep = “\_”, remove = TRUE)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| col | The name of the new column. |
| …. | A selection of desired columns. If empty, all variables are selected. |
| sep | A separator to use between values. |
| remove | If TRUE, remove input columns from output data frame. |

**Example:**

Unite is the compliment of separate. To undo **separate()**, we can use **unite()**, which merges two variables into one. Here we will merge two columns Group and Number with a separator**“.”**.

* R

|  |
| --- |
| # import tidyr package  library(tidyr)    long <- tidy\_dataframe %>%              gather(Group, Frequency,                     Group.1:Group.3)    # use separate() function to make data wider  separate\_data <- long %>%              separate(Group, c("Allotment",                                "Number"))    # use unite() function to glue  # Allotment and Number columns  unite\_data <- separate\_data %>%              unite(Group, Allotment,                    Number, sep = ".")    # print the new data frame  unite\_data |

**Output:**

S.No Group Frequency

1 1 Group.1 23

2 2 Group.1 345

3 3 Group.1 76

4 4 Group.1 212

5 5 Group.1 88

6 6 Group.1 199

7 7 Group.1 72

8 8 Group.1 35

9 9 Group.1 90

10 10 Group.1 265

11 1 Group.2 117

12 2 Group.2 89

13 3 Group.2 66

14 4 Group.2 334

15 5 Group.2 90

16 6 Group.2 101

17 7 Group.2 178

18 8 Group.2 233

19 9 Group.2 45

20 10 Group.2 200

21 1 Group.3 29

22 2 Group.3 101

23 3 Group.3 239

24 4 Group.3 289

25 5 Group.3 176

26 6 Group.3 320

27 7 Group.3 89

28 8 Group.3 109

29 9 Group.3 199

30 10 Group.3 56

* **spread() function:**It helps in reshaping a longer format to a wider format. The **spread()** function spreads a key-value pair across multiple columns.

***Syntax:***

*spread(data, key, value, fill = NA, convert = FALSE)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| key | Column names or positions. |
| value | Column names or positions. |
| fill | If set, missing values will be replaced with this value. |
| convert | If TRUE, type.convert() with asis = TRUE will be run on each of the new columns. |

**Example:**

We can transform the data from long back to wide with the **spread()** function.

* R

|  |
| --- |
| # import tidyr package  library(tidyr)    long <- tidy\_dataframe %>%              gather(Group, Frequency,                     Group.1:Group.3)    # use separate() function to make data wider  separate\_data <- long %>%              separate(Group, c("Allotment",                                "Number"))    # use unite() function to glue  # Allotment and Number columns  unite\_data <- separate\_data %>%              unite(Group, Allotment,                    Number, sep = ".")    # use unite() function to make data wider  back\_to\_wide <- unite\_data %>%              spread(Group, Frequency)    # print the new data frame  back\_to\_wide |

**Output:**

S.No Group.1 Group.2 Group.3

1 1 23 117 29

2 2 345 89 101

3 3 76 66 239

4 4 212 334 289

5 5 88 90 176

6 6 199 101 320

7 7 72 178 89

8 8 35 233 109

9 9 90 45 199

10 10 265 200 56

* **nest() function:**It creates a list of data frames containing all the nested variables. Nesting is implicitly a summarizing operation. This is useful in conjunction with other summaries that work with whole datasets, most notably models.

***Syntax:****nest(data, …, .key = “data”)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| …. | A selection of columns. If empty, all variables are selected. |
| .key | The name of the new column, as a string or symbol. |

**Example:**Let’s try to nest Group.2 column from the tidy\_dataframe we created in the data set.

* R

|  |
| --- |
| # import tidyr package  library(tidyr)    df <- tidy\_dataframe    # nest column Group.1 in  # tidy\_dataframe using nest()  df %>% nest(data = c(Group.1)) |

**Output:**

# A tibble: 10 x 4

S.No Group.1 Group.3 data

<int> <dbl> <dbl> <list>

1 1 23 29 <tibble [1 x 1]>

2 2 345 101 <tibble [1 x 1]>

3 3 76 239 <tibble [1 x 1]>

4 4 212 289 <tibble [1 x 1]>

5 5 88 176 <tibble [1 x 1]>

6 6 199 320 <tibble [1 x 1]>

7 7 72 89 <tibble [1 x 1]>

8 8 35 109 <tibble [1 x 1]>

9 9 90 199 <tibble [1 x 1]>

10 10 265 56 <tibble [1 x 1]>

* **unnest() function:**It basically reverses the nest operation. It makes each element of the list its own row. It can handle list columns that contain atomic vectors, lists, or data frames (but not a mixture of the different types).

***Syntax:***

*unnest(data, …, .drop = NA, .id = NULL, .sep = NULL, .preserve = NULL)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame |
| …. | Specification of columns to unnest.  If omitted, defaults to all list-columns. |
| .drop | Should additional list columns be dropped? By default,  it will drop them if unnesting  the specified columns requires the rows to be duplicated. |
| .id | Data frame identifier. |
| .sep | If non-NULL, the names of unnested data frame columns  will combine the name of the original list-col with  the names from nested data frame, separated by .sep. |
| .preserve | List-columns to preserve in the output. These will be   duplicated in the same way as atomic vectors. |

**Example:**

We will try to nest and unnest Species column in the iris dataframe in the tidyr package.

* R

|  |
| --- |
| # import the tidyr package  library(tidyr)    df <- iris  names(iris)    # nesting the species column in  # the df data frame using nest()  head(df %>% nest(data = c(Species)))  # Output (i)    # unnesting the species column  # in the df data frame using unnest()  head(df %>% unnest(Species,.drop = **NA**,                     .preserve = **NULL**)) # Output (ii) |

**Output (i):**

# A tibble: 6 x 5

Sepal.Length Sepal.Width Petal.Length Petal.Width data

<dbl> <dbl> <dbl> <dbl> <list>

1 5.1 3.5 1.4 0.2 <tibble [1 x 1]>

2 4.9 3 1.4 0.2 <tibble [1 x 1]>

3 4.7 3.2 1.3 0.2 <tibble [1 x 1]>

4 4.6 3.1 1.5 0.2 <tibble [1 x 1]>

5 5 3.6 1.4 0.2 <tibble [1 x 1]>

6 5.4 3.9 1.7 0.4 <tibble [1 x 1]>

**Output (ii):**

# A tibble: 6 x 5

Sepal.Length Sepal.Width Petal.Length Petal.Width Species

<dbl> <dbl> <dbl> <dbl> <fct>

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

5 5 3.6 1.4 0.2 setosa

6 5.4 3.9 1.7 0.4 setosa

* **fill() function:**Used to fill in the missing values in selected columns using the previous entry. This is useful in the common output format where values are not repeated, they’re recorded each time they change. Missing values are replaced in atomic vectors; NULL is replaced in the list.

***Syntax:***

*fill(data, …, .direction = c(“down”, “up”))*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| …. | A selection of columns. If empty, nothing happens. |
| direction | Direction in which to fill missing values. Currently, either “down” (the default) or “up” |

**Example:**

* R

|  |
| --- |
| # import the tidyr package  df <- data.frame(Month = 1:6,                   Year = c(2000, rep(**NA**, 5)))    # print the df data frame  df                   # Output (i)    # use fill() to fill missing values in  # Year column in df data frame  df %>% fill(Year)    # Output (ii) |

**Output (i):**

Month Year

1 1 2000

2 2 NA

3 3 NA

4 4 NA

5 5 NA

6 6 NA

**Output (ii):**

Month Year

1 1 2000

2 2 2000

3 3 2000

4 4 2000

5 5 2000

6 6 2000

* **full\_seq() function:**It basically fills the missing values in a vector which should have been observed but weren’t. The vector should be numeric.

***Syntax:****full\_seq(x, period, tol = 1e-06)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| x | A numeric vector. |
| period | Gap between each observation. |
| tol | Numerical tolerance for checking periodicity. |

**Example:**

* R

|  |
| --- |
| # import the tidyr package  library(tidyr)    # creating a numeric vector  num\_vec <- c(1, 7, 9, 14, 19, 20)    # use full\_seq() to fill missing  # values in num\_vec  full\_seq(num\_vector, 1) |

**Output:**

[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

* **drop\_na() function:**This function drops rows containing missing values.

***Syntax:****drop\_na(data, …)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| …. | A selection of columns. If empty, all variables are selected. |

**Example:**

* R

|  |
| --- |
| # import tidyr package  library(tidyr)    # create a tibble df with missing values  df <- tibble(S.No = c(1:10),               Name = c('John', 'Smith', 'Peter',                        'Luke', 'King', rep(**NA**, 5)))    # print df tibble  df                    # Output (i)    # use drop\_na() to drop columns  # in df with missing values  df %>% drop\_na(Name)  # Output (ii) |

**Output (i):**

# A tibble: 10 x 2

S.No Name

<int> <chr>

1 1 John

2 2 Smith

3 3 Peter

4 4 Luke

5 5 King

6 6 <NA>

7 7 <NA>

8 8 <NA>

9 9 <NA>

10 10 <NA>

**Output (ii):**

# A tibble: 5 x 2

S.No Name

<int> <chr>

1 1 John

2 2 Smith

3 3 Peter

4 4 Luke

5 5 King

* **replace\_na() function:**Itreplaces missing values.

***Syntax:****replace\_na(data, replace, …)*

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| data | A data frame. |
| replace | If data is a data frame, returns a data frame. If data is a vector,  returns a vector of class determined by the union of data and replace. |

**Example:**

* R

|  |
| --- |
| # import tidyr package  library(tidyr)    df <- data.frame(S.No = c(1:10),                   Name = c('John', 'Smith',                            'Peter', 'Luke',                            'King', rep(**NA**, 5)))    df                                      # Output (i)    # use replace\_na() to replace missing values or na  df %>% replace\_na(list(Name = 'Henry')) # Output (ii) |

**Output (i):**

# A tibble: 10 x 2

S.No Name

<int> <chr>

1 1 John

2 2 Smith

3 3 Peter

4 4 Luke

5 5 King

6 6 <NA>

7 7 <NA>

8 8 <NA>

9 9 <NA>

10 10 <NA>

**Output (ii):**

S.No Name

1 1 John

2 2 Smith

3 3 Peter

4 4 Luke

5 5 King

6 6 Henry

7 7 Henry

8 8 Henry

9 9 Henry

10 10 Henry

# What Are the Tidyverse Packages in R Language?

When dealing with Data Science in the [**R Programming Language**](https://www.geeksforgeeks.org/r-programming-language-introduction/), the Tidyverse packages are your best friends! These Tidyverse packages were specially designed for Data Science with a common design philosophy.

They include all the [packages](https://www.geeksforgeeks.org/packages-in-r-programming/) required in the data science workflow, ranging from data exploration to data visualization. For example, readr is for data importing, tibble and tidyr help in tidying the data, dplyr and stringr contribute to data transformation and ggplot2 is vital for data visualization.



## Tidyverse Packages in R

There are eight core Tidyverse packages namely**ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr,** and **forcats** that are mentioned in this article. All of these packages are loaded automatically at once with the install.packages(“tidyverse”) command.

In addition to these packages, Tidyverse also has some specialized packages that are not loaded automatically but need their own call. These include the DBI for relational databases. httr for web APIs, rvest for web scraping, etc. Now, let’s see the core Tidyverse packages and learn more about them.

### Tidyverse Packages in R following:

1. Data Visualization and Exploration
   * ggplot2
2. Data Wrangling and Transformation
   * dplyr
   * tidyr
   * stringr
   * forcats
3. Data Import and Management
   * tibble
   * readr
4. Functional Programming
   * purrr

## Data Visualization and Exploration in Tidyverse in R

### 1. ggplot2:

[ggplot2](https://www.geeksforgeeks.org/using-ggplot2-package-in-r-programming/) is an R data visualization library that is based on The Grammar of Graphics. ggplot2 can create data visualizations such as bar charts, pie charts, histograms, scatterplots, error charts, etc. using high-level API. It also allows you to add different types of data visualization components or layers in a single visualization.

Once ggplot2 has been told which variables to map to which aesthetics in the plot, it does the rest of the work so that the user can focus on interpreting the visualizations and take less time in creating them. But this also means that it is not possible to create highly customized graphics in ggplot2. But there are a lot of resources in the RStudio community and Stack Overflow which can provide help in ggplot2 when needed.

If you want to install ggplot2, the best method is to install the tidyverse using:

install.packages("tidyverse")

Or you can just install ggplot2 using:

install.packages("ggplot2")

You can also install the development version from GitHub using:

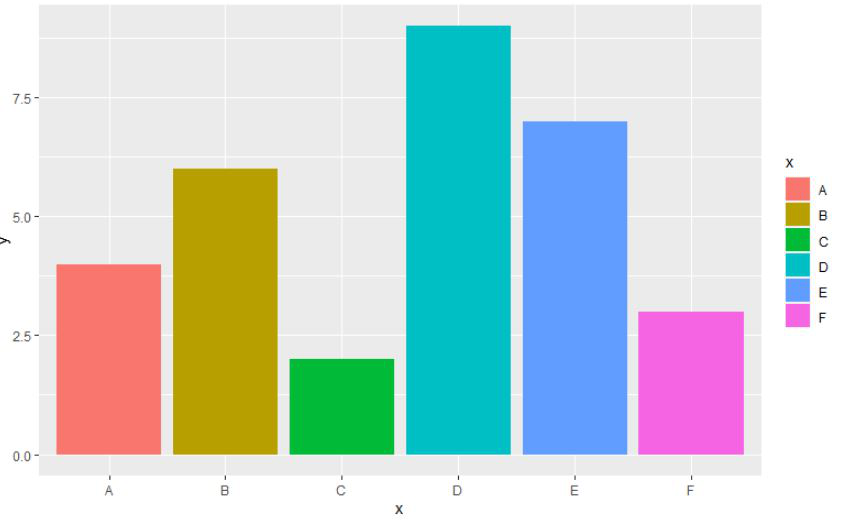
devtools::install\_github("tidyverse/ggplot2")

We will be using 6 different data points for the bar plot and then with the help of the fill argument within the aes function, we will be applying the default colors to the barplot in the R programming language.

* R

|  |
| --- |
| # load the library  library("ggplot2")    # create the dataframe with letters and numbers  gfg <-data.frame(      x=c('A', 'B', 'C', 'D', 'E', 'F'),      y=c(4, 6, 2, 9, 7, 3))    # display the bar  ggplot(gfg, aes(x, y, fill=x)) + geom\_bar(stat="identity") |

**Output:**



## Data Wrangling and Transformation in Tidyverse in R

### 1. dplyr:

[dplyr](https://www.geeksforgeeks.org/dplyr-package-in-r-programming/)is a very popular data manipulation library in R. It has five important functions that are combined naturally with the group\_by() function that can help in performing these functions in groups.

These functions include the mutate() function which can add new variables that are functions of existing variables, select() function that selects the variables based on their names, filter() function that picks selects the variables based on their values.

summarise() function that reduces multiple values into a summary, and the arrange() function that arranges the arranges the row orderings.  If you want to install dplyr, the best method is to install the tidyverse using:

install.packages("tidyverse")

Or you can just install dplyr using:

install.packages("dplyr")

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/dplyr")

* R

|  |
| --- |
| library(dplyr)    print(starwars %>% filter(species == "Droid")) |

**Output:**

# A tibble: 6 x 14  
 name height mass hair\_color skin\_color eye\_color birth\_year sex gender   
 <chr> <int> <dbl> <chr> <chr> <chr> <dbl> <chr> <chr>   
1 C-3PO 167 75 <NA> gold yellow 112 none masculi~  
2 R2-D2 96 32 <NA> white, blue red 33 none masculi~  
3 R5-D4 97 32 <NA> white, red red NA none masculi~  
4 IG-88 200 140 none metal red 15 none masculi~  
5 R4-P17 96 NA none silver, red red, blue NA none feminine  
6 BB8 NA NA none none black NA none masculi~  
# ... with 5 more variables: homeworld <chr>, species <chr>, films <list>,  
# vehicles <list>, starships <list>

### 2. tidyr:

[tidyr](https://www.geeksforgeeks.org/tidyr-package-in-r-programming/) is a data cleaning library in R which helps to create tidy data. Tidy data means that all the data cells have a single value with each of the data columns being a variable and the data rows being an observation.

This tidy data is a staple in the tidyverse and it ensures that more time is spent on data analysis and to obtain value from data rather than cleaning the data continuously and modifying the tools to handle untidy data.

The functions in tidyr broadly fall into five categories namely, Pivoting which changes the data between long and wide forms, Nesting which changes grouped data so that a group is a single row with a nested data frame, Splitting character columns and then combining them, Rectangling which converts nested lists into tidy tibbles and converting implicit missing values into explicit values. If you want to install tidyr, the best method is to install the tidyverse using:

install.packages("tidyverse")

Or you can just install tidyr using:

install.packages("tidyr")

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/tidyr")

The gather() function in tidr will take multiple columns and collapse them into key-value pairs, duplicating all other columns as needed.

* R

|  |
| --- |
| # load the tidyr package  library(tidyr)    n = 10  # creating a data frame  tidy\_dataframe = data.frame(                      S.No = c(1:n),                      Group.1 = c(23, 345, 76, 212, 88,                                  199, 72, 35, 90, 265),                      Group.2 = c(117, 89, 66, 334, 90,                              101, 178, 233, 45, 200),                      Group.3 = c(29, 101, 239, 289, 176,                                  320, 89, 109, 199, 56))    # print the elements of the data frame  print(head(tidy\_dataframe))      # using gather() function on tidy\_dataframe  long <- tidy\_dataframe %>%              gather(Group, Frequency,                  Group.1:Group.3)    # print the data frame in a long format  print(head(long)) |

**Output:**

S.No Group.1 Group.2 Group.3  
1 1 23 117 29  
2 2 345 89 101  
3 3 76 66 239  
4 4 212 334 289  
5 5 88 90 176  
6 6 199 101 320  
print the data frame in a long format  
 S.No Group Frequency  
1 1 Group.1 23  
2 2 Group.1 345  
3 3 Group.1 76  
4 4 Group.1 212  
5 5 Group.1 88  
6 6 Group.1 199

### 3. Stringr:

[stringr](https://www.geeksforgeeks.org/stringr-package-in-r-programming/) is a library that has many functions used for data cleaning and data preparation tasks. It is also designed for working with strings and has many functions that make this an easy process. stringr is built on top of stringi, which is an International Components for Unicode C library.

So if there are any functions that you want to use but cannot find in stringr, then the best place to look for them is stringi. This also means that once you master stringr, stringi is not that difficult to use as both of these packages have similar conventions.

All of the functions in stringr start with str and they take a string vector as their first argument. Some of these functions include str\_detect(), str\_extract(), str\_match(), str\_count(), str\_replace(), str\_subset(), etc.  If you want to install stringr, the best method is to install the tidyverse using:

install.packages("tidyverse")

Or you can just install stringr from CRAN using:

install.packages("stringr")

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/stringr")

* R

|  |
| --- |
| # R program for finding length of string    # Importing package  library(stringr)    # Calculating length of string  str\_length("hello") |

**Output:**

5

### 4. Forcats:

[forcats](https://www.geeksforgeeks.org/forcats-package-in-r-programming/)is a R library that is concerned with handling problems associated with vectors. These vectors are variables that have a fixed set of possible values they can take which is already known in advance. So forecast deals with issues like changes the orders of values in vectors, reordering the vectors, etc.

Some of the functions in forcats are fct\_relevel() that reorders a vectors by hand, fct\_reorder() that reorders a factor using another variable, fct\_infreq() that reorders a factorby frequency values etc. If you want to install forcats, the best method is to install the tidyverse using:

install.packages("tidyverse")

Or you can just install forcats from using:

install.packages("forcats")

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/forcats")

* R

|  |
| --- |
| library(forcats)  library(dplyr)  library(ggplot2)    print(head(starwars %>% filter(!is.na(species))             %>% count(species, sort = **TRUE**))) |

**Output:**

# A tibble: 6 x 2  
 species n  
 <chr> <int>  
1 Human 35  
2 Droid 6  
3 Gungan 3  
4 Kaminoan 2  
5 Mirialan 2  
6 Twi'lek 2

## Data Import and Management in Tidyverse in R

### 1. readr:

This readr library provides a simple and speedy method to read rectangular data such as that with file formats tsv, csv, delim, fwf, etc. readr can parse many different types of data using a function that parses the total file and another that focuses on the specific column.

This column specification defines the method to convert the data in the column from a character vector to the data type that is most suited. This is done automatically by readr in most cases.

readr can read different kinds of file formats using different functions, namely read\_csv() for comma-separated files, read\_tsv() for tab-separated files, read\_table() for tabular files, read\_fwf() for fixed-width files, read\_delim() for delimited files, and, read\_log() for web log files. If you want to install readr, the best method is to install the tidyverse using:

install.packages("tidyverse").

Or you can just install readr using:

install.packages("readr").

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/readr")

Reading file with readr library.

* R

|  |
| --- |
| # R program to read text file  # using readr package    # Import the readr library  library(readr)    # Use read\_tsv() to read text file  myData = read\_tsv("geeksforgeeks.txt", col\_names = **FALSE**)  print(myData) |

**Output:**

# A tibble: 1 x 1  
 X1   
   
1 A computer science portal for geeks.

### 2. tibble:

A [tibble](https://www.geeksforgeeks.org/data-wrangling-in-r-programming-working-with-tibbles/" \t "_blank)is a form of a data.frame which includes the useful parts of it and discards the parts that are not so important. So tibbles don’t change variables names or types like data.frames nor do they do partial matching but they bring problems to the forefront much sooner such as when a variable does not exist.

So a code with tibbles is much cleaner and effective than before. Tibbles is also easier to use with larger datasets that contain more complex objects, in part before an enhanced print() method.

You can create new tibbles from column vectors using the tibble() function and you can also create a tibble row-by-row using a tribble() function. If you want to install tibble, the best method is to install the tidyverse using:

install.packages("tidyverse"):

Or you can just install tibble using:

install.packages("tibble")

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/tibble")

* R

|  |
| --- |
| library(tibble)  data <- data.frame(a = 1:3, b = letters[1:3],                     c = Sys.Date() - 1:3)  print(data) |

**Output:**

a b c  
1 1 a 2021-11-24  
2 2 b 2021-11-23  
3 3 c 2021-11-22

## Functional Programming in Tidyverse in R

### 1. purrr:

[Purrr](https://www.geeksforgeeks.org/purrr-package-in-r-programming/)is a detailed set of tools for functions and vectors and it is mainly used to manage the functional programming in R.

A good example of this is the map() functions that are used to replace multiple for loops that complicate and mess up the code into simpler code that is easy to read. In addition to that, all purrr functions are type-stable

which means they either return the advertised output type and if that is not possible, then the give an error. If you want to install purrr, the best method is to install the tidyverse using:

install.packages("tidyverse")

Or you can just install purrr using:

install.packages("purrr")

You can also install the development version from GitHub using:

devtools::install\_github("tidyverse/purrr")

* R

|  |
| --- |
| library(purrr)    mtcars %>%    split(.$cyl) %>% # from base R    map(~ lm(mpg ~ wt, data = .)) %>%    map(summary) %>%    map\_dbl("r.squared") |

**Output:**

4 6 8   
0.5086326 0.4645102 0.4229655