Manipulating Strings With *glue*

Lona Dai

12/02/2017

Introduction

In homework 4, we made use of cat( ), paste( ), format( ) …. to display strings in the ways we wanted. But often, to get some simple string outputs, we have to use some or all of these functions, which requires long and tedious coding. Is there an easier way to manipulate strings? Fortunately, yes! In this post, I will introduce a new package called *glue* which glues strings together in a faster way and only uses a single function.

Get Ready?

Firstly, we need to install the required package for *glue*. Use install.packages("devtools") in console. I also typed in install.packages("glue") in console to make sure I'm having the updated package.

After installing, we can load in glue and start our work.

```
library(glue)
```

```
## Warning: package 'glue' was built under R version 3.4.2
```

Real Data Examples

Let's look at an example from the r documentation on how glue works.

Ex.1 glue( )

```
name <- "Fred"
age <- 50
anniversary <- as.Date("1991-10-12")
glue('My name is {name},',
  ' my age next year is {age + 1},',
  ' my anniversary is {format(anniversary, "%A, %B %d, %Y")}.')
```

```
## My name is Fred, my age next year is 51, my anniversary is Saturday, October 12, 1991.
```

We can see in glue( ), you can actually call the previous created objects in a string by using { }, and manipulate those objects in { } to display the strings in correct formats.

We can also change the contents of the objects but not change the structure of the sentence.

```
name <- "Mike"
glue('My name is {name},',
  ' my age next year is {age + 1},',
  ' my anniversary is {format(anniversary, "%A, %B %d, %Y")}.')
```

```
## My name is Mike, my age next year is 51, my anniversary is Saturday, October 12, 1991.
```

With glue( ), we can combine multiple functions into a single function with a nice and clean format.

I will create a 3 by 3 simple data frame *df* to do the following illustration with glue( ).

```
df <- data.frame("Character" = c("one", "two", NA), "Numeric" = 1:3, "Logical" = c(TRUE, TRUE, FALSE), stringsAsFa
ctors = FALSE)

df
```

```
##   Character Numeric Logical
## 1       one       1    TRUE
## 2       two       2    TRUE
## 3      <NA>       3   FALSE
```

```
msg <- 'Dataframe Info: \n\n This dataset has {nrow(df)} rows and {ncol(df)} columns. \n There {ifelse(sum(is.na(d
f))>0,"is","are")} {sum(is.na(df))} Missing Value'

glue(msg)
```

```
## Dataframe Info:
##
## This dataset has 3 rows and 3 columns.
## There is 1 Missing Value
```

From the above example, we can see that we can even write functions in { }.

Instead of using *backslah n*, We can also start a new line like this

```
msg <- 'Dataframe Info:

This dataset has {nrow(df)} rows and {ncol(df)} columns.
There {ifelse(sum(is.na(df))>0,"is","are")} {sum(is.na(df))} Missing Value'

glue(msg)
```

```
## Dataframe Info:
##
## This dataset has 3 rows and 3 columns.
## There is 1 Missing Value
```

In addition, if you do not want to make a new line in your string, you can use double backslah:

```
msg <- 'Dataframe Info: \\
This dataset has {nrow(df)} rows and {ncol(df)} columns. \\
There {ifelse(sum(is.na(df))>0,"is","are")} {sum(is.na(df))} Missing Value'

glue(msg)
```

```
## Dataframe Info: This dataset has 3 rows and 3 columns. There is 1 Missing Value
```

EX.2 glue_data( )

Usually, we do not only want to look at the basic information about a data frame but the content in the data frame. What should we do? There is another function called glue_data( ) which can access the data in the data frame and manipulate strings like what we did with glue( ).

IMPORTANT NOTE: glue_data( ) function needs %>% operator from *magrittr*, thus, let's load in *magrittr* first.

```
library(magrittr)
```

Still use the same data frame created above for this illustration.

```
df <- data.frame("Character" = c("one", "two", NA), "Numeric" = 1:3, "Logical" = c(TRUE, TRUE, FALSE), stringsAsFa
ctors = FALSE)

df %>% glue_data("{Character} equals {Numeric} is {Logical}")
```

```
## one equals 1 is TRUE
## two equals 2 is TRUE
## NA equals 3 is FALSE
```

EX.3 glue_sql( )

When we write SQL queries in database, it can always cause error because of quatation syntax. To avoid such unnecessary mistakes, I will introduce glue_sql( ) which handles SQL quoting. If you are unfamiliar with SQL, I will suggest this link to look up (usefulSQLResource). However, since the concepts I will introduce here are not very much related to the format of SQL, it is ok to proceed for now to learn the concepts and usages of glue_sql( ) first if you do not want to spend time learning SQL.

First, load in the packages we need for SQL (Don't forget to install.packages("DBI"), install.packages("RSQLite")).

```
library(DBI)
library(RSQLite)
```

We can get access to the database provided in the package.

```
con <- dbConnect(RSQLite::SQLite(), ":memory:")
colnames(iris) <- gsub("[.]", "_", tolower(colnames(iris)))
DBI::dbWriteTable(con, "iris", iris)
```

Now, let's make use of glue_sql( ).

```
var <- "sepal_width"
tbl <- "iris"
num <- 2
val <- "setosa"
glue_sql("
  SELECT {`var`}
  FROM {`tbl`}
  WHERE {`tbl`}.sepal_length > {num}
    AND {`tbl`}.species = {val}
  ", .con = con)
```

```
## <SQL> SELECT `sepal_width`
## FROM `iris`
## WHERE `iris`.sepal_length > 2
##   AND `iris`.species = 'setosa'
```

We can see that since we can set the quatation marks in { } before inputing the string data, we can avoid the mistakes that will easily be made in quotation syntax.

Because in the future, we will be dealing with lots of data which will be stored in database, it is useful to know glue_sql( ) function to help you manage your data, especially strings, more easily without making syntax error.

Summary

In this post, I introduced a new package called *glue* that is helpful for string manipulation. Three functions in this package I think are the most useful ones when dealing with strings. The glue( ) function can concatenate strings together in one single function without having multiple functions like cat( ), paste( ), format( ), etc. The glue_data( ) function allows us to get access to data frames, but it requires %>% operator from another package called *magrittr*. The glue_sql( ) function makes it easy to create or access strings in database by avoiding unnecessary quotatioin syntax errors.

Take Home Message

If you need to manipulate strings but do not want to waste time debugging your code because of various functions involved, definitely try *glue*, the easiest and simplest function for string manipulation.

Reference

1. https://www.rdocumentation.org/packages/glue/versions/1.2.0/topics/glue
2. https://www.r-bloggers.com/creating-reporting-template-with-glue-in-r/
3. https://awesome-r.com/
4. https://cran.r-project.org/web/packages/glue/glue.pdf
5. https://github.com/tidyverse/glue/issues/59
6. http://rpubs.com/jimhester/glue_sql_example
7. https://github.com/tidyverse/glue
8. https://datascienceplus.com/creating-reporting-template-with-glue-in-r/