R Workshop

Lecture 5: More advanced functionality in R and R Studio, Part II

May 15, 2024

1 A useful function for concatenating strings into a longer string

Sometimes, you want to create a string that is actually composed of other strings. Maybe, you want to create new strings for file names in a for loop. `paste()` is a very useful function for this.

```
#Creating is a single new string
newString = paste("Hello","World")
print(newString)

## [1] "Hello World"

#Note, a file extension is also a string, so you can add that too
newString2 = paste("Hello","World",".png")
print(newString2)

## [1] "Hello World .png"

#In the above cases, there is a space between the two strings.
#This is because there is a "sep" argument with a space as the default.
#Below, I made the sep value equal to no space
newString3 = paste("Hello","World",".png",sep="")
print(newString3)

## [1] "HelloWorld.png"

#Now, let's try to iterate the process
for(i in 1:10){
  newString = paste("Filename",i,".png",sep="")
  print(newString)
}

## [1] "Filename1.png"
## [1] "Filename2.png"
## [1] "Filename3.png"
## [1] "Filename4.png"
## [1] "Filename5.png"
## [1] "Filename6.png"
## [1] "Filename7.png"
## [1] "Filename8.png"
## [1] "Filename9.png"
## [1] "Filename10.png"
```

2 Opening multiple files

We already saw last time that we can concatenate multiple data files using the `bind_rows()` function. However, we manually specified the file names. We will now try to automate the process. Let's assume that all your data files are `.csv` and are in a single sub-folder in your Project. We can then find out what the vector of `.csv` file names in that sub-folder is, and then use that vector in a for loop, and open each file separately.

To get the vector of `.csv` files in a folder, you need to use the `list.files()` function. If you print the variable, you will see a list of all the file names in the relevant sub-folder.

```
fileNames = list.files(path="Results/")
print(fileNames)

##  [1] "ExperimentResults_1.csv"  "ExperimentResults_10.csv"
##  [3] "ExperimentResults_11.csv" "ExperimentResults_12.csv"
##  [5] "ExperimentResults_13.csv" "ExperimentResults_14.csv"
##  [7] "ExperimentResults_15.csv" "ExperimentResults_16.csv"
##  [9] "ExperimentResults_17.csv" "ExperimentResults_18.csv"
## [11] "ExperimentResults_19.csv" "ExperimentResults_2.csv"
## [13] "ExperimentResults_20.csv" "ExperimentResults_21.csv"
## [15] "ExperimentResults_22.csv" "ExperimentResults_23.csv"
## [17] "ExperimentResults_24.csv" "ExperimentResults_25.csv"
## [19] "ExperimentResults_26.csv" "ExperimentResults_27.csv"
## [21] "ExperimentResults_28.csv" "ExperimentResults_29.csv"
## [23] "ExperimentResults_3.csv"  "ExperimentResults_30.csv"
## [25] "ExperimentResults_31.csv" "ExperimentResults_32.csv"
## [27] "ExperimentResults_33.csv" "ExperimentResults_34.csv"
## [29] "ExperimentResults_35.csv" "ExperimentResults_36.csv"
## [31] "ExperimentResults_37.csv" "ExperimentResults_38.csv"
## [33] "ExperimentResults_39.csv" "ExperimentResults_4.csv"
## [35] "ExperimentResults_40.csv" "ExperimentResults_5.csv"
## [37] "ExperimentResults_6.csv"  "ExperimentResults_7.csv"
## [39] "ExperimentResults_8.csv"  "ExperimentResults_9.csv"
```

Now, we can use this list of names to open each file and then bind them. Remember, in the for loop, you can give it *any* vector — R cycles through all the positions in the vector; it's not really incrementing a counter as in other programming languages.

```
#Creating a blank data.frame
#This is needed so that you can keep updating the data.frame
DataFull = NULL

for(i in fileNames){

  #Opening a single data file
  data = read.csv(i)

  #Opening each file and then concatenating it
  DataFull = DataFull %>%
    bind_rows(i)
}

## Warning in file(file, "rt"): cannot open file 'ExperimentResults_1.csv': No such file
or directory
## Error in file(file, "rt"): cannot open the connection
```

```
#Viewing the new data.frame
head(DataFull)

## NULL
```

The above code results in an error because we gave `read.csv()` just the file name and not the sub-directory information. We need to give it both the directory and file name as a single argument. So, we need to use `paste()` to create a file name that includes the sub-directory information, and then pass that to `read.csv()`.

```
#Creating a blank data.frame
#This is needed so that you can keep updating the data.frame
DataFull = NULL

for(i in fileNames){
  #create the string
  FileNameAndAddress = paste("Results/",i,sep="")

  #Opening a single data file
  data = read.csv(FileNameAndAddress)

  #Opening each file and then concatenating it
  DataFull = DataFull %>%
    bind_rows(data)
}

#Viewing the new data.frame
head(DataFull)

##   Sub Condition Measurement
## 1   1         A -0.81441721
## 2   2         B  1.16779592
## 3   3         A  0.58212838
## 4   4         B  0.09095144
## 5   5         A -1.01658201
## 6   6         B -0.03351326

#How many rows does it have
nrow(DataFull)

## [1] 4000
```

If you notice, *DataFull* has all the data from all the files in it, but the `data.frame` doesn't have any information about the source of the information. Sometimes, it is useful to know which file a set of data comes from — especially if you have a separate file for participant or condition or group. To achieve this, you need to create another column in each `data.frame` before binding it.

```
#Creating a blank data.frame
#This is needed so that you can keep updating the data.frame
DataFull = NULL

for(i in fileNames){
  #create the string
  FileNameAndAddress = paste("Results/",i,sep="")
  #Opening a single data file
  data = read.csv(FileNameAndAddress) %>%
    mutate(FileName = i)

  #Opening each file and then concatenating it
  DataFull = DataFull %>%
    bind_rows(data)
}

#Viewing the new data.frame
head(DataFull)

##   Sub Condition Measurement                FileName
## 1   1         A -0.81441721 ExperimentResults_1.csv
## 2   2         B  1.16779592 ExperimentResults_1.csv
## 3   3         A  0.58212838 ExperimentResults_1.csv
## 4   4         B  0.09095144 ExperimentResults_1.csv
## 5   5         A -1.01658201 ExperimentResults_1.csv
## 6   6         B -0.03351326 ExperimentResults_1.csv

#How many rows does it have
nrow(DataFull)

## [1] 4000
```

3 Changing the shape of your data

Sometimes, you have the data you want to plot in separate columns, but the plot functions allow you to plot only a single column. In such cases, you have to first change the layout of your `data.frame` and then plot it. Two functions that are useful are `pivot_longer()` and `pivot_longer()`.

`pivot_longer()` can be used to take two columns and put all the values in a single column — it takes a `data.frame` that is wider and makes it longer; hence, the name of the function. The long format is particularly useful to work with in R and `tidyverse` functions.

```
#Creating the data
Data1 = data.frame(Sub = 1:100,
                   Year1 = rnorm(100, mean=100,sd=10))
Data1$Year2 = Data1$Year1+rnorm(100, mean=10,sd=20)


#Warping the data into a long format
Data1_long = Data1 %>%
  pivot_longer(cols=Year1:Year2, names_to="Year", values_to="Values")
head(Data1_long)

## # A tibble: 6 x 3
##     Sub Year  Values
##   <int> <chr>  <dbl>
## 1     1 Year1  109.
## 2     1 Year2   98.9
## 3     2 Year1  124.
## 4     2 Year2  110.
## 5     3 Year1   99.6
## 6     3 Year2  123.

# tail(Data1_modified)
```

Notice, what happened above, the values of the two separate columns *Year1* and *Year2* were collapsed into a new column called *Values*, and another column *Year* was created to mark whether the value is from the original *Year1* or *Year2*.

You can also use `pivot_wider()` do warp the `data.frame` in the opposite way — make a long format into a wide format.

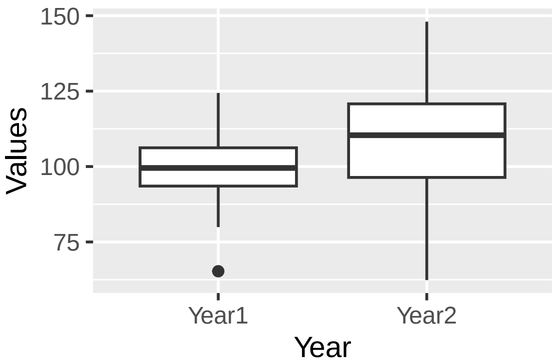
```
#Warping the data into a wide format
Data1_wide = Data1_long %>%
  pivot_wider(names_from="Year", values_from="Values")
head(Data1_wide)

## # A tibble: 6 x 3
##     Sub Year1 Year2
##   <int> <dbl> <dbl>
## 1     1 109.   98.9
## 2     2 124.  110.
## 3     3  99.6 123.
## 4     4 108.  113.
## 5     5  86.4  95.9
## 6     6  95.9  97.2
```

4 Integrating data munging and plotting

Below, I use a slightly different style of referring to the `data.frame` than I have done in the past — this way you can truly integrate your data analysis code with your plotting code.

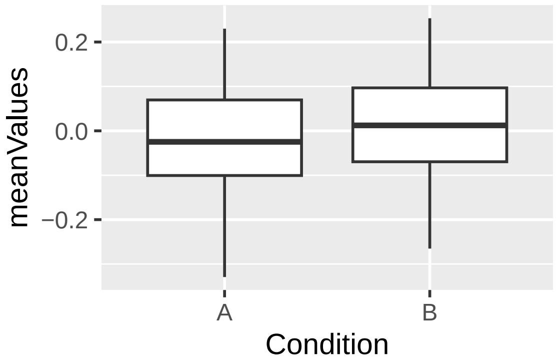
```
Data1_long %>%
  ggplot(aes(x=Year, y=Values))+
  geom_boxplot()
```



So, you can perform multiple steps of data munging and then plot that modified data without saving it to a `data.frame`. For this, I will use the *DataFull* that we created above in Section 2.

```
DataFull %>%
  group_by(FileName,Condition) %>%
  summarise(meanValues = mean(Measurement)) %>%
  ggplot(aes(x=Condition, y=meanValues))+
  geom_boxplot()

## `summarise()` has grouped output by 'FileName'. You can override using the
## `.groups` argument.
```


meanValues
A
B
Condition