Post01: dplyr VS. SQL: Which Do I Use, and When?

Vibha Seshadri

October 23, 2017

Introduction

Data Manipulation. *Data Analytics*. *Data* _____ . Let's just say *data* is a huge buzzword these days. But data is more than just a buzzword. As more and more companies and organizations focus on data collection, manipulation, and analysis, more people are learning how to use different Data related tools. I am currently in STAT133, a course here at UC Berkeley which focuses on Concepts in Computing with Data. A few weeks ago, we learned about a handy package in R created by Hadley Wickham called `dplyr`. `dplyr` provides functions which one can use in replacement of base R indexing and subsetting when analyzing and manipulating data frames. The best part is that `dplyr` is easy to use, easy to learn, and extremely powerful! But when should one use `dplyr` as opposed to a querying language such as `SQL`, when working with data? This post aims to explore the pros and cons of both `dplyr` and `SQL` and explain when one should be used instead of the other.

Motivation

Why did I choose this topic?

I decided to write a post comparing `dplyr` and `SQL` because when I first started learning about `dplyr`, I was curious about when `SQL` was more appropriate than `dplyr` and vice versa. I wondered this was because I had heard a lot about `SQL` and had even used it briefly in another class, but I had never heard of, or used `dplyr`. After learning how to use `dplyr` I was baffled as to why anyone would ever use `SQL`! `dplyr` was easier to learn and use, so I decided to research the differences and similarities between the two, so in the future, I would know in which scenarios I could use `dplyr` over `SQL` and vice versa.

Background

So exactly what exactly are `SQL` and `dplyr`?

Before we can answer that question, let's define two terms.

Data Set A data set is a collection of related discrete data that can be accessed by a user and manipulated or read. The term data set was invented by IBM to mean something similar to a file. Data sets are stored in data structures, such as a database (read below), or CSV file.

Database A database is a type of *data structure*. It can store tables of information on a computer. There are many forms of Databases: Hierarchical databases, Object Oriented Databases, Network Databases, and Relational Databases. SQL is used with relational databases. You can access the tables in a relational database and the contents of these tables using a querying language. You can also create new tables from the tables in the relational database, and manipulate the data in these tables using a querying languages. Almost all companies store multitudes of company financial and employee data in secure databases. Databases are also used for the backend of many websites to store data gathered from or about website users because it allows easy manipulation and use of the data on the website. Earlier, most databases were "flat" meaning they could only be used in a manner similar to how spreadsheets are used today. Luckily, today, we have Database Management Systems which allow more complex and secure querying of databases, sometimes even between multiple databases! Modern databases also allow us to store media such as pictures and videos in databases as opposed to just text and numeric values.

Now that we understand how data can be stored and reprented, let's talk about `SQL` and `dplyr`!

Let's start with `SQL`. To learn about `SQL` I did some research on W3Schools, a great website for anyone who wants to teach themselves a new programming language! I also watched a youtube video. After reading what I had to learn, you should do some of your own research! It's quite fun!

`SQL` stands for Strucutred Querying Language. It is mainly used with RDMS's such as mySQL, Oracle, MS SQL Server, Microsoft Access, etc. `SQL` allows you to insert, search, update, delete, join, filter, aggregate, select, etc. in your *database* to organize, update, query, and access your data. You can even create new databases using `SQL` and set permissions on who can and can't access certain databses. This latter part is especially important in helping companies keep their employee, financial, legal, etc data secure. There are many different versions of the `SQL` language, and some companies even have their own prorietary version of `SQL` The different versions of `SQL` vary from just syntax differences to what the language can do overall. `SQL` can be used in the back end of a website to store data that needs to be displayed or collected. In order to actually connect to a RDMS that is queried using `SQL`, you need to access you database using server side scripts such as `PHP` or `ASP`.

Now that we know a little bit about `SQL` let's learn about `dplyr`! After we learn about `dplyr` we will compare and contrast `SQL` and `dplyr`.

`dplyr` is an `R` package created by Hadley Wickham. `dplyr` is a great tool for manipulating *data sets*. It's fast and connects very easily and well with languages like `SQL`, and `Google biquery`. `dplyr` also allows you to work with *large* databases without having to import the database into R since `dplyr` can connect with databases such as PostgreSQL. Prior to `dplyr`, all datasets had to be imported into R to be used in data manipulation. Additionally, `dplyr` provides individual functions to handle things like filtering, aggregating, selecting, summarising, etc. Each of these functions only have on individual function but they can be easily connected together by using `%>%`, the piping operator which takes the output from one function call and applies it to the next. Additionally, `dplyr` doesn't change the contents of your data, it only manipulates it. So you need to save your work to a new data frame or reassign to your data frame the output of the data manipulation.

Compare and Contrast

OK I get it, but what's different between `dplyr` and `SQL`?

Plain and simple, `dplyr` is meant to be used for data analysis and `SQL` is meant to be used when storing, managing, and querying large data sets. Use `SQL` to organize and query your data. Use `dplyr` to analyze your data. `SQL` was not created or commercialized as a data analysis tool, it was created and commercialized as a data management tool. Additionally, `R` when combined with `dplyr` makes data analysis cleaner and easier. What takes 10 to 15 lines in `SQL` can take 1 or 2 lines in `dplyr`.

Examples

Let's see some `dplyr` in action!

Since `dplyr` is meant for data analysis let's see how it can be used!

We will be using the dataset `UCBAdmissions` included in the `R` `datasets` package. This dataset contains the "Aggregate data on applicants to graduate school at Berkeley for the six largest departments in 1973 classified by admission and sex."

Let's learn a little bit about the data before we begin manipulating it.

```
# Save the data in a R object
cal <- UCBAdmissions;

# Check if the data is in a dataframe, if it isn't then make it one
if (!class(cal) == "data.frame") {
  cal <- as.data.frame.table(cal, stringsAsFactors = FALSE)
}

# Explore the structure of the data
str(cal)
```

```
## 'data.frame':    24 obs. of  4 variables:
##  $ Admit : chr  "Admitted" "Rejected" "Admitted" "Rejected" ...
##  $ Gender: chr  "Male" "Male" "Female" "Female" ...
##  $ Dept  : chr  "A" "A" "A" "A" ...
##  $ Freq  : num  512 313 89 19 353 207 17 8 120 205 ...
```

```
# Dimensions of the data
nrow(cal)
```

```
## [1] 24
```

```
ncol(cal)
```

```
## [1] 4
```

```
# Column and row names of data set
colnames(cal)
```

```
## [1] "Admit"  "Gender" "Dept"   "Freq"
```

```
rownames(cal)
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20" "21" "22" "23" "24"
```

```
# First few elements of the data
head(cal, n = 5);
```

```
##      Admit Gender Dept Freq
## 1 Admitted   Male    A  512
## 2 Rejected   Male    A  313
## 3 Admitted Female    A   89
## 4 Rejected Female    A   19
## 5 Admitted   Male    B  353
```

```
# Summary statistics of the data
summary(cal)
```

```
##     Admit              Gender              Dept                Freq
##  Length:24          Length:24          Length:24          Min.   :  8.0
##  Class :character   Class :character   Class :character   1st Qu.: 80.0
##  Mode  :character   Mode  :character   Mode  :character   Median :170.0
##                                                           Mean   :188.6
##                                                           3rd Qu.:302.5
##                                                           Max.   :512.0
```

Let's see how many total male graduate appliclants were admitted to UC Berkeley in 1973, and how many total female graduate applicants were

admitted to UC Berkeley in 1973 using `dplyr`.

```
# Total Male Accepted
male <- cal %>% filter(Gender == "Male") %>%
  filter(Admit == "Admitted") %>%
  select(Freq) %>%
  summarise(sum(Freq))

# Total Female Accepted
female <- cal %>% filter(Gender == "Female") %>%
  filter(Admit == "Admitted") %>%
  select(Freq) %>%
  summarise(sum(Freq))

# How many more men than women were accepted?
male - female
```

```
##   sum(Freq)
## 1       641
```

To do something similar in SQL we would have had to had written something along the following lines.

```
Declare @mSum int
Declare @fSum int
select sum(freq) as @mSum from cal where Gender = 'Male' and Admit = 'Admitted';
select sum(freq) as @fSum from cal where Gender = 'Male' and Admit = 'Admitted;
@mSum - @fSum
```

You may be wondering, well that's not all that bad. Well, in the previous example we were just querying the data, or retrieving information provided to us in the dataset. We weren't really performing data analysis. That's why the `SQL` equivalent didn't seem so bad. So let's ask again, why exactly is `dplyr` a better choice for data analysis again?

Imagine a scenario in which we would want to calculate the percentages of males and females accepted into and rejected from the graduate program in 1973, and then graph the results that we find. Well this is where `dplyr` is perfect! While much of what we may want to do to get these percentages can be done in `SQL` too, the fact that `dplyr` is an `R` package makes it simple for us to perform the analysis and to use other `R` packages such as `ggplot2` to graph and illustrate our findings. To graph the results we find from using `SQL`, we would have to ensure the environment we are using while coding in `SQL` has the capabilities to graph data. Even then, doing so is not as simple and elegant as it is in `R`. Because using `dplyr` is the better choice for the following example, the analysis and calculations for the question posed above is done in `dplyr`.

```
# Function which calculates percentage of applicants accepted in DEPT
calAdmit <- function (df, dept) {
  accept <- (df$Freq[df$Admit == 'Admitted' & df$Dept == dept])
  total <- sum(df$Freq[df$Dept == dept])
  accept/total
}

# Function which calculates percentage of applicants rejected in DEPT
calReject <- function (df, dept) {
  reject <- (df$Freq[df$Admit == 'Rejected' & df$Dept == dept])
  total <- sum(df$Freq[df$Dept == dept])
  reject/total
}

# Creating data frame of percentages of males accepted and rejected by Dept
males <- cal %>% filter(Gender == 'Male')
mP <- males %>% group_by(Dept) %>%
  summarise(Accepted = calAdmit(males, Dept),
            Rejected = calReject(males, Dept))

# Creating data frame of percentages of females accepted and rejected by Dept
females <- cal %>% filter(Gender == 'Female')
fP <- females %>% group_by(Dept) %>%
  summarise(Accepted = calAdmit(females, Dept),
            Rejected = calReject(females, Dept))

# Percentages of males accepted and rejected by dept
mP
```

```
## # A tibble: 6 x 3
##    Dept   Accepted  Rejected
##   <chr>      <dbl>     <dbl>
## 1     A 0.62060606 0.3793939
## 2     B 0.63035714 0.3696429
## 3     C 0.36923077 0.6307692
## 4     D 0.33093525 0.6690647
## 5     E 0.27748691 0.7225131
## 6     F 0.05898123 0.9410188
```

```
# Percentages of females accepted and rejected by dept
fP
```

```
## # A tibble: 6 x 3
##    Dept   Accepted  Rejected
##   <chr>      <dbl>     <dbl>
## 1     A 0.82407407 0.1759259
## 2     B 0.68000000 0.3200000
## 3     C 0.34064081 0.6593592
## 4     D 0.34933333 0.6506667
## 5     E 0.23918575 0.7608142
## 6     F 0.07038123 0.9296188
```

Let's see what percentage of males were *accepted* into the different departments compared to the percentage of females accepted.

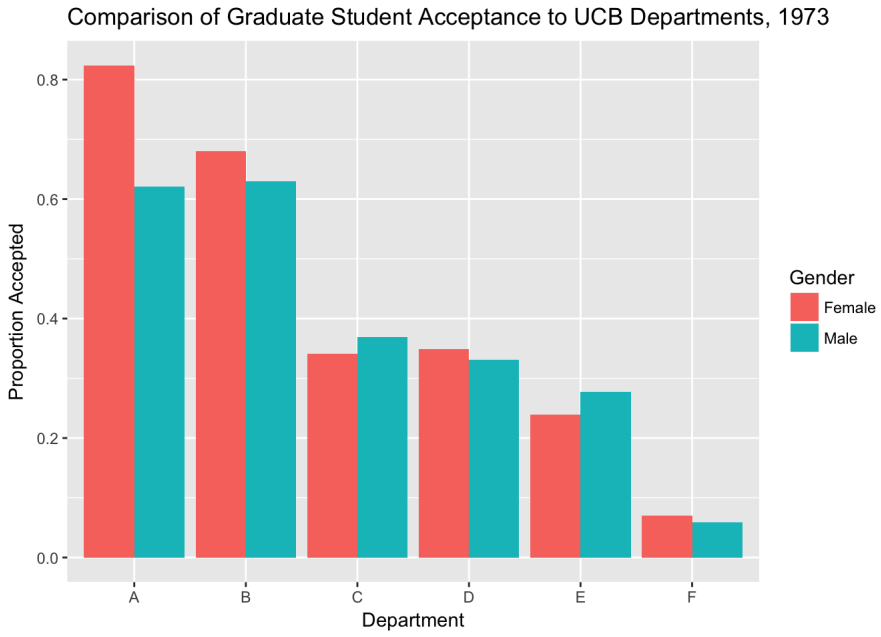
```
### Joining the tables that hold the percentages of males and females accepted
### and rejected into different departments

df.list <- list(a=mP, b=fP)
accept <- stack(lapply(df.list, `[[`, "Accepted"))
dept <- stack(lapply(df.list, `[[`, "Dept"))
joined <- data.frame(Department = as.character(dept$values),
                     Accepted = as.numeric(accept$values),
                     Gender = as.character(dept$ind),
                     stringsAsFactors = FALSE)
joined$Gender[joined$Gender == 'a'] <- "Male"
joined$Gender[joined$Gender == 'b'] <- "Female"

# contents of 'joined'
joined
```

```
##    Department   Accepted Gender
## 1           A 0.62060606   Male
## 2           B 0.63035714   Male
## 3           C 0.36923077   Male
## 4           D 0.33093525   Male
## 5           E 0.27748691   Male
## 6           F 0.05898123   Male
## 7           A 0.82407407 Female
## 8           B 0.68000000 Female
## 9           C 0.34064081 Female
## 10          D 0.34933333 Female
## 11          E 0.23918575 Female
## 12          F 0.07038123 Female
```

```
# Plotting the data in the joined table
ggplot(joined, aes(x = Department, y = Accepted, fill = Gender)) +
  geom_bar(position="dodge", stat = "Identity") +
  ggtitle("Comparison of Graduate Student Acceptance to UCB Departments, 1973") +
  ylab("Proportion Accepted")
```



Conclusions

So what's the take away?

Both `dplyr` and `SQL` are great tools to be familiar with if you are interested in data; however, it's important to understand the distnction of when both should be used. Use `SQL` if you want to organize, store, and query your data. Use `dplyr` for any data analysis you may want to perform.

References

- How I made the SQL and dplyr images at the top render nicely
- Background information on Data Sets
- Background information on Databases
- What is a Database and SQL?
- W3Schools
- Introductory information on dyplr
- Four Reasons You Should Check Out the R package dyplr
- Why SQL is not for analysis, but dyplr is
- How I created the side by side bar chart