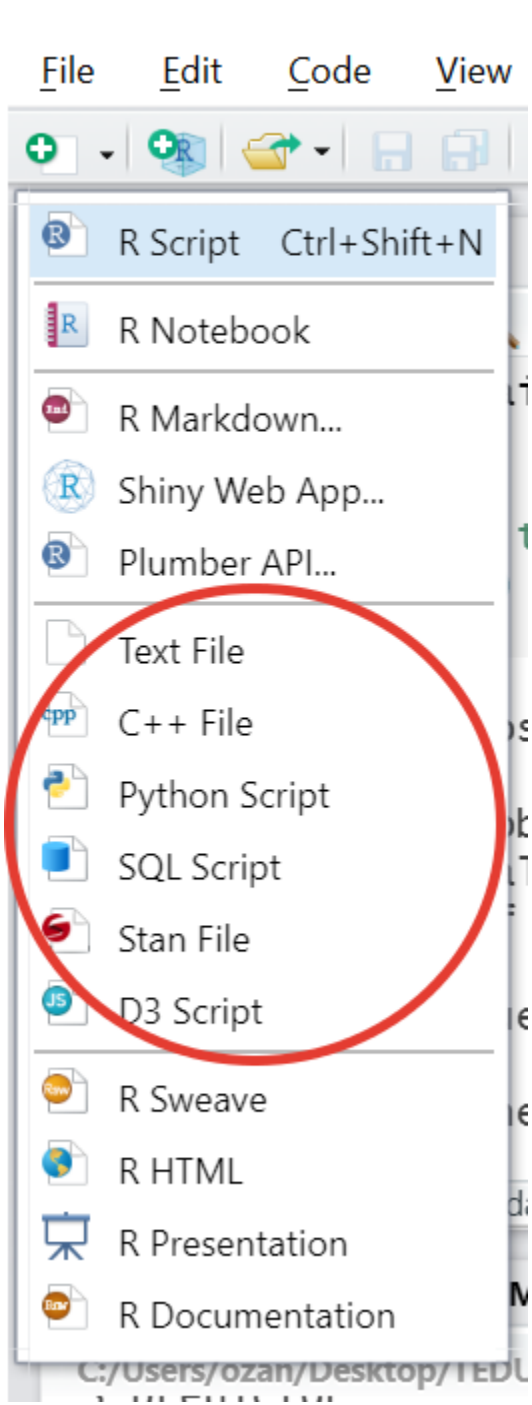
Ve`R`i tabanları ile kısa konuşmalar

ozanevkaya
Nov 22 · 4 min read

RIP dear Jim Henson

R dilinde daha başka neler yapılabilir noktasında sınırları kestirmek zor olsa da, **diller arası iletişim** açısından oldukça iyi durumda olduğunu söylemek herhalde yanlış olmaz. Malumunuz, **Python ve daha nicesini** çalıştırmak mümkün. Örneğin, yeni bir script açmak istediğimizde başımıza gelebilecekler adına,

RStudio


File
Edit
Code
View
R Script Ctrl+Shift+N
R Notebook
R Markdown...
Shiny Web App...
Plumber API...
Text File
C++ File
Python Script
SQL Script
Stan File
D3 Script
R Sweave
R HTML
R Presentation
R Documentation

Bir gece çılgınlığı neticesinde, R ortamında veri tabanları ile kurabileceğimiz ilişkiye örnek olması adına, **R ortamında SQL (Structured Query Language)** tarafına göz atalım derim. Konuyla ilgili olarak kısaca **SQLDF** ve **RSQLite** paketlerine göz kırpıyor olacağız.

1. **SQLDF**

Yazarlarının deyimiyle

The sqldf() function is typically passed a single argument which is an SQL select statement where the table names are ordinary R data frame names. sqldf() transparently sets up a database, imports the data frames into that database, performs the SQL select or other statement and returns the result using a heuristic to determine which class to assign to each column of the returned data frame. The sqldf() or read.csv.sql() functions can also be used to read filtered files into R even if the original files are larger than R itself can handle. 'RSQLite', 'RH2', 'RMySQL' and 'RPostgreSQL' backends are supported.

Opps, bir de **'RH2', 'RMySQL' and 'RPostgreSQL'** çıktı ortaya, neyse sakin kalmaya çalışmalı ! Ana hatlarıyla **alternatifler** kısaca şöyle

- DBI
- RODBC
- dbConnect
- RSQLite

- RMySQL
- RPostgreSQL

Biz yine **SQLDF** içerisinde neler var ona bakmaya çalışalım. Paket içerisinde sadece `sqldf()` fonksiyonu bulunmakta. Özellikle **SQL dili**nde yer alan anahtar kelimeleri kullanarak verileri düzenlemek vb. oldukça kolay görünüyor.

```
> # put a condition
> sqldf("SELECT * FROM EuStock WHERE DAX > 1620 LIMIT 5")
     DAX    SMI    CAC   FTSE
1 1628.75 1678.1 1772.8 2443.6
2 1621.04 1684.1 1708.1 2470.4
3 1630.75 1682.9 1734.5 2487.9
4 1640.17 1703.6 1757.4 2508.4
5 1635.47 1697.5 1754.0 2510.5
> # Ordering
> sqldf('SELECT * FROM EuStock ORDER BY CAC ASC LIMIT 5')
     DAX    SMI    CAC   FTSE
1 1402.34 1788.0 1611.0 2446.3
2 1421.49 1820.5 1612.5 2488.4
3 1542.77 1598.3 1633.6 2345.4
4 1548.44 1603.1 1636.9 2392.0
5 1543.99 1591.5 1645.6 2380.2
> # Aggregated data
> sqldf("SELECT AVG(DAX) FROM EuStock")
  AVG(DAX)
1 2530.657
> 
```

Tabi bu kısım sadece verinin manipülasyonuna dair, herhangi bir şekilde veri tabanına bağlanma durumu görünür de yok. Şimdi biraz da o tarafa göz atmaya çalışalım, yoksa bu üsttekini dplyr ya da base paketleri de yapmak varken, ne zorumuza !

2. **RSQLite**

Super-hero Hadley'den alıntıyla;

RSQLite is the easiest way to use a database from R because the package itself contains SQLite; no external software is needed.

Yine ondan bir not, aslında **DBI paketinde yer alan built-in fonksiyonları** kullanıyoruz;

RSQLite is a DBI-compatible interface which means you primarily use functions defined in the DBI package, so you should always start by loading DBI, not RSQLite

---

Örneğin **bir database yaratarak oraya veriyi aktarmak ve SQL dilinde işlemler yapmak mümkün**. Burada sorgu göndermek açısından temel fonksiyon `dbGetQuery()`

```
36
37 # To create a new SQLite database
38 mydb <- dbConnect(RSQLite::SQLite(), "my-dbsql")
39 dbDisconnect(mydb)
40 unlink("my-dbsql")
41
42 # For a temporary database it can be used the following
43 mydb <- dbConnect(RSQLite::SQLite(), "")
44 dbDisconnect(mydb)
45
46 # Copying a data set into new database
47 mydb <- dbConnect(RSQLite::SQLite(), "my-dbsql")
```

```
dbWriteTable(mydb, "EuStock", EuStock)
dbListTables(mydb)
```

```
Console  Terminal  Jobs
C:/Users/ozan/Desktop/TEDU_ADA 442 Statistical Learning/Lecture Notes-Slides/Week 6/
> dbGetQuery(mydb, 'SELECT * FROM EuStock WHERE "DAX" > :x LIMIT 10', params = list(x = 1620))
      DAX    SMI    CAC   FTSE
1  1628.75 1678.1 1772.8 2443.6
2  1621.04 1684.1 1708.1 2470.4
3  1630.75 1682.9 1734.5 2487.9
4  1640.17 1703.6 1757.4 2508.4
5  1635.47 1697.5 1754.0 2510.5
6  1645.89 1716.3 1754.3 2497.4
7  1647.84 1723.8 1759.8 2532.5
8  1638.35 1730.5 1755.5 2556.8
9  1629.93 1727.4 1758.1 2561.0
10 1621.49 1733.3 1757.5 2547.3
>
```

Source: SQL_R script

Herhangi bir şekilde veri setleri global environment içerisinde olmadan onları başka bir yere göndermiş ve üzerinde filtreleme yapmış olduk basit bir şekilde. Özellikle yüksek boyutlu veriler için bu tarz yöntemlerin avantajı malum. Burada biz RSQLite paketinden söz ettik ancak kapsayıcı bir özet olması açısından şu görsel durumu gayet iyi özetliyor;

- Establish database connection: package ***DBI = Database Interface***
- Use the appropriate backend package

Five common **backends**:

Backend Package	Database(s)
RMySQL	*MySQL* and *MariaDB*
RPostgreSQL	*Postgres* and *Redshift*
RSQLite	*SQLite*
odbc	Many commercial DBs (*open database connectivity protocol*)
bigquery	Google's *BigQuery* (hosted database for big data)

bigrquery	Google's *BigQuery* (hosted database for big data)

DBI has several **vignettes**, see *help(package = "DBI")*

Source: https://github.com/fjodor/data-import-export/blob/main/Accessing-SQL.pdf

Bu arada ilk görsele tekrar dönmek gerekirse, Rmarkdown dosyası içerisinde tıpkı R kodu yazıp işlem yapar gibi SQL için de benzer bir yöntem sözkonusu, bakınız aşağıdaki görsel;

R Notebooks support SQL code chunks like so:

```
```{sql, connection=con, output.var = "mydataframe"}
SELECT "month_idx", "year", "month",
SUM(CASE WHEN ("term_deposit" = 'yes')
THEN (1.0) ELSE (0.0) END) AS "subscribe",
COUNT(*) AS "total" FROM ("bank")
GROUP BY "month_idx", "year", "month"
```
```

Chunk options:
- *max.print = 10* for number of records displayed (set to -1 or NA for no limit)
- tab.cap = "My Caption": caption indicates number of records displayed

- Markdown documents enable you to use different languages within the same file
- Just like specifying an R code block, you can also specify an sql code block using **```{sql,**
- Requires you to first establish a connection (here: **con**), usually via *DBI::dbConnect()*
- Benefit: syntax highlighting
- Find out more about language engines supported by R Markdown: https://bookdown.org/yihui/rmarkdown/language-engines.html

Source: https://github.com/fjodor/data-import-export/blob/main/Accessing-SQL.pdf

Bir parantez de **dbplyr paketi** için açmak gerekiyor. Özellikle **dplyr** grammerine alışkın olanlar açısında hızla uyum sağlanabilecek bir yol diyebiliriz. Yine yazarlarının deyimiyle,

---

A 'dplyr' back end for databases that allows you to work with remote database tables as if they are in-memory data frames. Basic features works with any database that has a 'DBI' back end; more advanced features

require 'SQL' translation to be provided by the package author.

Verilerin R hafızasında olup olmadığına bakmaksızın işlem yapmak mümkün görünüyor. Örneğin aynı veriyi, benzer şekilde oluşturduğumuz geçici veri tabanına aktarmak,

```
> # Copying data to con
> copy_to(con, EuStock, "EuStock",
+         temporary = FALSE,
+         indexes = list("DAX", "SMI", "CAC", "FTSE") )
> # Put a reference on it
> EuStock_db <- tbl(con, "EuStock")
> EuStock_db
# Source:   table<EuStock> [?? x 4]
# Database: sqlite 3.36.0 [:memory:]
     DAX   SMI   CAC  FTSE
   <dbl> <dbl> <dbl> <dbl>
 1 1629. 1678. 1773. 2444.
 2 1614. 1688. 1750. 2460.
 3 1607. 1679. 1718  2448.
 4 1621. 1684. 1708. 2470.
 5 1618. 1687. 1723. 2485.
 6 1611. 1672. 1714. 2467.
 7 1631. 1683. 1734. 2488.
 8 1640. 1704. 1757. 2508.
 9 1635. 1698. 1754  2510.
10 1646. 1716. 1754. 2497.
# ... with more rows
```

Source: SQL_R script

ve arkasından **dplyr diline benzer şekilde işlemler** yaparak bir nevi **SQL sorgularını** gerçekleştirmek mümkün;

```
# Some queries like dplyr syntax
EuStock_db %>% select(DAX)

EuStock_db %>% filter(DAX>1620)

EuStock_db %>%
  select(DAX, SMI) %>%
  filter(DAX>1620) %>%
  head()
```

Source: SQL_R script

Nereler ile iletişim kurulabilir kısmına gelirsek, dbplyr paketi şunlara imkan tanıyor;

- Oracle
- Microsoft SQL Server
- PostgreSQL
- Amazon Redshift
- Apache Hive
- Apache Impala

Her ne kadar çok kısa bir giriş olsa da bakılması gereken esas yer sanıyorum burası.

https://db.rstudio.com/

Veri tabanları ile iletişim kurmak konusunda gerçekten kurcalamam gereken bir yer diye düşünüyorum. Siz ne dersiniz ?

Elde ettiğimiz **çıktıları yeniden üretmek**, farklı biçimlerde kullanmak için **gerekli kodlar** ve ile ilgili kaynaklara **https://github.com/oevkaya/** adresinden erişebiliriniz.

Görüş, öneri ve takipte kalmak için: https://www.linkedin.com/in/ozanevkaya/ https://twitter.com/ozanevkaya