MariaDB with R

Dr. Zahid Ansari

Table of Contents

1 MariaDB with R 1
1.1 Preparing and importing data into a local MariaDB database 1
1.2 Working with MariaDB from RStudio 6

1 MariaDB with R

- In this section, we will query a MariaDB database installed on a local, personal computer directly from RStudio.

```
setwd("D:/AMU Computer Science/Courses/Big Data Analytics/Big Data Analytics Using R/Ch5")
```

1.1 Preparing and importing data into a local MariaDB database

- Download MariaDB from https://downloads.mariadb.org/
- For me the downloaded installer file was mariadb-10.11.0-winx64.msi
- To install MariaDB on Windows, you follow these steps:
- Step 1. Double-click the installer to start the installation process.
- Step 2. Accept the end-user license agreement
- Step 3. Select features: Choose the directory that stores the MariaDB files and click the Next button. The default location on Windows is C:Files
- Step 4. Set root's password: Type a password for the root user account. You will use this password to connect to MariaDB later. The root user is the default user of the MariaDB, which has all privileges.
- If you don't want the root user to login from a remote machine, you need to uncheck the Enable access from remote machines for 'root' user checkbox.
- The Use UTF8 as the default server's character set option allows you to use the UTF8 as the default character set when you create new databases and tables.
- Once you complete selecting all options, click the Next button to go to the next step.

- Step 5. Configure Database: In this step First, install MariaDB as a service by selecting the Install as service option. It allows you to rename the service name.
- Second, configure the port for the MariaDB. By default, MariaDB uses 3306 port. However, you can change it to your port if you want.
- Third, specify the parameters for the Innodb engine including buffer pool size and page size. 16KB page size is suitable for most databases.
- Finally, click the Next button to go to the next step.
- Step 6. Submit usage information: If you want to submit anonymous usage information so that MariaDB developers can improve the system, check the checkbox and click the Next button
- Step 7. Ready to install MariaDB: Click the Install button to start installing MariaDB
- Step 8. Complete the MariaDB setup: Click the Finish button to complete MariaDB setup
- You can find the MariaDB tools in the startup menu:
- Click on MySQL Client (MariaDB 10.11 (x64)) from start menu and provide the root password to get the MardiDB command prompt
- Provide a previously specified password for the root user to authorize the login. If identified correctly, you should now see a welcome message.
- Once you have installed MariaDB on a Windows. Therefore, there are only a few simple tasks ahead of us before we can start thinking of how to connect MariaDB with the R environment:

1. Create a database and a table that will hold our data.
2. Move the data from the data file directory to the database.

- MariaDB is largely based on the MySQL framework and as such it operates in a very similar way.
- Upon logging in to MariaDB, you can view all available databases to the root user:

```
MariaDB [(none)]> SHOW databases;
```

Database |

information_schema
mysql |
performance_schema

3 rows in set (0.00 sec)

- We can create a new database named data1 using the CREATE command:

```
MariaDB [(none)]> CREATE database data1;
```

Query OK, 1 row affected (0.00 sec)

```
MariaDB [(none)]> SHOW databases;
```

Database |
data1 |
information_schema
mysql |
performance_schema

4 rows in set (0.01 sec)

- We will now select the data1 database, in which we will create a new table called need:

```
MariaDB [(none)]> USE data1
```

Database changed MariaDB [data1]>

- As you can see, the MariaDB command prompt now includes (in the square brackets) the reference to the selected database data1. You can double-check the selection at any time using the following line:

```
MariaDB [data1]> SELECT database();
```

database()
data1 |

1 row in set (0.00 sec)

- The data1 database is currently empty, as confirmed by the output of the SHOW tables; command:

```
MariaDB [data1]> SHOW tables;
```

Empty set (0.00 sec)

- We will now create a new table called need, which will store the data according to the defined schema:

```
MariaDB [data1]> CREATE TABLE need(
  hh_id INTEGER,
  region VARCHAR(25),
  imd_eng VARCHAR(25),
  imd_wales VARCHAR(25),
  gcons2005 VARCHAR(25),
  gcons2005valid VARCHAR(25),
  gcons2006 VARCHAR(25),
  gcons2006valid VARCHAR(25),
  gcons2007 VARCHAR(25),
  gcons2007valid VARCHAR(25),
  gcons2008 VARCHAR(25),
  gcons2008valid VARCHAR(25),
```

```
  gcons2009 VARCHAR(25),
  gcons2009valid VARCHAR(25),
  gcons2010 VARCHAR(25),
  gcons2010valid VARCHAR(25),
  gcons2011 VARCHAR(25),
  gcons2011valid VARCHAR(25),
  gcons2012 VARCHAR(25),
  gcons2012valid VARCHAR(25),
  econs2005 VARCHAR(25),
  econs2005valid VARCHAR(25),
  econs2006 VARCHAR(25),
  econs2006valid VARCHAR(25),
  econs2007 VARCHAR(25),
  econs2007valid VARCHAR(25),
  econs2008 VARCHAR(25),
  econs2008valid VARCHAR(25),
  econs2009 VARCHAR(25),
  econs2009valid VARCHAR(25),
  econs2010 VARCHAR(25),
  econs2010valid VARCHAR(25),
  econs2011 INTEGER,
  econs2011valid VARCHAR(25),
  econs2012 VARCHAR(25),
  econs2012valid VARCHAR(25),
  e7flag2012 VARCHAR(25),
  main_heat_fuel INTEGER,
  prop_age INTEGER,
  prop_type INTEGER,
  floor_area_band INTEGER,
  ee_band INTEGER,
  loft_depth INTEGER,
  wall_cons INTEGER,
  cwi VARCHAR(25),
  cwi_year VARCHAR(25),
  li VARCHAR(25),
  li_year VARCHAR(25),
  boiler VARCHAR(25),
  boiler_year VARCHAR(25));
```

Query OK, 0 rows affected (0.02 sec)

- In the preceding call, we have indicated the data types for each variable in the data.
- The output informs us that zero rows were affected, as we have only created the structure (schema) for our data, with no data read in yet.
- You can inspect the schema of the need table within the data1 database, using the DESCRIBE command:

```
MariaDB [data1]> DESCRIBE need;
```

Field	Type	Null	Key	Default	Extra

… #output truncated

- We can now upload the data to the need table we created.
- As the data is stored in the /home/swalko/ directory, we will first have to create a new user for the database with all privileges for reading and writing from/to this database.
- To keep it simple and transparent, our new user will be called swalko and its access will be identified by a password, for example Password1:

```
MariaDB [data1]> CREATE USER 'ansari'@'localhost' IDENTIFIED BY 'ansari';
```

Query OK, 0 rows affected (0.00 sec)

- We will then grant all privileges to swalko on the newly created table within the data1 database:

```
MariaDB [data1]> GRANT ALL PRIVILEGES ON data1.need TO
'ansari'@'localhost' IDENTIFIED BY "ansari" with grant option;
```

Query OK, 0 rows affected (0.00 sec)

- Re-load all privileges to activate them:

```
MariaDB [data1]> FLUSH PRIVILEGES;
```

Query OK, 0 rows affected (0.00 sec)

- And log out from MariaDB as the root user

```
MariaDB [data1]> EXIT
```

Bye

```
C:\Program Files\MariaDB 10.11\bin>mysql -P 3316 -u ansari -pansari
```

Welcome to the MariaDB monitor. Commands end with ; or . Your MariaDB connection id is 12 Server version: 10.11.0-MariaDB mariadb.org binary distribution Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others. Type ‘help;’ or ‘ for help. Type’ to clear the current input statement. MariaDB [(none)]>

- Check whether you have access to the need table by executing a number of commands:

```
MariaDB [(none)]> SHOW databases;
```

Database |

data1 |
information_schema

2 rows in set (0.00 sec)

```
MariaDB [(none)]> USE data1
```

Database changed

```
MariaDB [data1]> SHOW tables;
```

Tables_in_data1

need |

1 row in set (0.00 sec)

```
MariaDB [data1]> DESCRIBE need;
```

Field Type Null Key Default Extra

… #output truncated 50 rows in set (0.00 sec)

- All seems to be working just fine. We may now use the data stored in the need_puf_2014.csv and upload it to the need table:

```
MariaDB [data1]> LOAD DATA LOCAL INFILE
'D:/AMU Computer Science/Courses/Big Data Analytics/Big Data Analytics Using
R/Ch5/need_puf_2014.csv'
INTO TABLE need
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Query OK, 49815 rows affected (1.27 sec) Records: 49815 Deleted: 0 Skipped: 0 Warnings: 0

- The output confirms that all 49,815 records have been copied successfully to the need table in the data1 database.
- We can now exit MariaDB and log out from the instance as the ansari user:

```
MariaDB [data1]> EXIT
```

Bye

1.2 Working with MariaDB from RStudio

- For the transfer of data between MariaDB Server and R Environment, it is recommended R's "odbc" Package: CRAN odbc
- For installing the "odbc" package from CRAN, execute in R:

```
install.packages("odbc")
```

- For installing RMariaDB package through CRAN, execute the following R statement:

```
install.packages("RMariaDB")
```

As we have already installed the RMariaDB package, we just need to load it into the R session.

```
library(RMariaDB)
```

- In order to be able to query the database, we first have to create a connection to the data1 database in MariaDB:

```
conn <- dbConnect(
  drv = RMariaDB::MariaDB(),
  user = "ansari",
  password = "ansari",
  host = "localhost",
  port = 3316,
  dbname = "data1")
```

- The preceding code connects R with MariaDB.We can obtain confirmation of the created connection with the summary() function:

```
summary(conn)
```

```
##            Length             Class              Mode
##                 1 MariaDBConnection                S4
```

The dbGetInfo() function provides a little bit more detailed output about the connection with MariaDB:

```
dbGetInfo(conn)
```

```
## $host
## [1] "localhost"
##
## $username
## [1] "ansari"
##
## $dbname
## [1] "data1"
##
## $con.type
## [1] "localhost via TCP/IP"
##
## $db.version
## [1] "10.11.0-MariaDB"
##
## $port
## [1] NA
##
## $protocol.version
## [1] 10
##
## $thread.id
## [1] 11
```

- We can now obtain the names of the tables present in the database using the dbListTables() function:

```
dbListTables(conn)
```

```
## [1] "need"
```

- We can now obtain the fields of the tables present in the database (that is, variables) using dbListFields() function:

```
dbListFields(conn, "need")
```

```
##  [1] "hh_id"           "region"          "imd_eng"         "imd_wales"
##  [5] "gcons2005"       "gcons2005valid"  "gcons2006"
"gcons2006valid"
##  [9] "gcons2007"       "gcons2007valid"  "gcons2008"
"gcons2008valid"
## [13] "gcons2009"       "gcons2009valid"  "gcons2010"
"gcons2010valid"
## [17] "gcons2011"       "gcons2011valid"  "gcons2012"
"gcons2012valid"
## [21] "econs2005"       "econs2005valid"  "econs2006"
"econs2006valid"
## [25] "econs2007"       "econs2007valid"  "econs2008"
"econs2008valid"
## [29] "econs2009"       "econs2009valid"  "econs2010"
"econs2010valid"
## [33] "econs2011"       "econs2011valid"  "econs2012"
"econs2012valid"
## [37] "e7flag2012"      "main_heat_fuel"  "prop_age"        "prop_type"
## [41] "floor_area_band" "ee_band"         "loft_depth"      "wall_cons"
## [45] "cwi"             "cwi_year"        "li"              "li_year"
## [49] "boiler"          "boiler_year"
```

- Let's test whether we can perform the simplest query by calculating the total number of records in our need table:

```
query.1 <- dbSendQuery(conn, "SELECT COUNT(*) AS records FROM need")
```

The standard RMySQL functions, such as dbGetStatement(), dbColumnInfo(), and dbGetInfo(), are also supported for MariaDB:

```
dbGetStatement(query.1)
```

```
## [1] "SELECT COUNT(*) AS records FROM need"
```

```
dbColumnInfo(query.1)
```

```
##      name      type
## 1 records integer64
```

```
dbGetInfo(query.1)
```

```
## $statement
## [1] "SELECT COUNT(*) AS records FROM need"
##
## $row.count
## [1] 0
```

```
##
## $rows.affected
## [1] 0
##
## $has.completed
## [1] FALSE
```

- We can pull the results of the query into R as a data.frame object using the dbFetch() function:

```
query.1.res <- dbFetch(query.1, n=-1)
query.1.res
```

```
##   records
## 1   49815
```

- As usual, after obtaining the aggregated or processed data from the database, we need to free the resources associated with a result set by running the dbClearResult() function:

```
dbClearResult(query.1)
```

- Let's perform the second query on the need table in the data1 database in MariaDB. We will calculate the average electricity consumption in 2012 (Econs2012) grouped by electricity efficiency band (ee_band), property age (prop_age), and property type (prop_type). We will sort the results in ascending order by first the electricity efficiency band and then property type:

```
query.2 <- dbSendQuery(conn, "SELECT EE_BAND, PROP_AGE, PROP_TYPE,
  AVG(Econs2012) AS AVERAGE_ELEC_2012
  FROM need
  GROUP BY EE_BAND, PROP_AGE, PROP_TYPE
  ORDER BY EE_BAND, PROP_TYPE ASC")
```

- The dbColumnInfo() function can again provide us with some useful information about the expected structure of the result set:

```
dbColumnInfo(query.2)
```

```
##                name    type
## 1           EE_BAND integer
## 2          PROP_AGE integer
## 3         PROP_TYPE integer
## 4 AVERAGE_ELEC_2012  double
```

- We may now fetch the results in the standard way:

```
query.2.res <- dbFetch(query.2, n=-1)
query.2.res
```

```
##     EE_BAND PROP_AGE PROP_TYPE AVERAGE_ELEC_2012
## 1         1      102       101         12162.500
## 2         1      106       101          5246.774
## 3         1      101       101          2650.000
## 4         1      104       101          4200.000
```

```
## 5          1       105       101          3933.333
## 6          1       103       101          3137.500
## 7          1       102       102          4775.000
## 8          1       101       102          3330.000
## 9          1       105       102          3366.667
## 10         1       104       102          4013.158
## 11         1       106       102          2979.545
## 12         1       103       102          3551.429
## 13         1       101       103          2350.000
## 14         1       106       103          3369.355
## 15         1       103       103          3465.000
## 16         1       102       103          3625.000
## 17         1       104       103          5225.000
## 18         1       105       103          4500.000
## 19         1       106       104          3439.109
## 20         1       104       104          3735.000
## 21         1       101       104          2410.000
## 22         1       103       104          3200.000
## 23         1       102       104          3835.714
## 24         1       105       104          2100.000
## 25         1       101       105          5116.667
## 26         1       106       105          4493.333
## 27         1       103       105          2354.167
## 28         1       104       105          2329.545
## 29         1       105       105          3078.571
## 30         1       103       106          1907.273
## 31         1       104       106          1814.115
## 32         1       106       106          2781.313
## 33         1       101       106          1900.000
## 34         1       105       106          2038.398
## 35         1       102       106          1493.750
## 36         2       101       101          7317.857
## 37         2       105       101          4537.674
## 38         2       104       101          4550.721
## 39         2       103       101          5320.253
## 40         2       106       101          4933.985
## 41         2       102       101          6286.735
## 42         2       105       102          3444.946
## 43         2       106       102          3421.673
## 44         2       103       102          3543.911
## 45         2       104       102          3462.408
## 46         2       101       102          4210.000
## 47         2       102       102          3703.739
## 48         2       101       103          4090.789
## 49         2       106       103          3397.479
## 50         2       105       103          2871.284
## 51         2       102       103          3720.787
## 52         2       104       103          3335.307
## 53         2       103       103          3479.319
## 54         2       102       104          3191.667
```

```
## 55       2      106       104          3418.889
## 56       2      101       104          3274.908
## 57       2      105       104          3031.366
## 58       2      104       104          3448.405
## 59       2      103       104          3595.195
## 60       2      102       105          3920.000
## 61       2      106       105          3938.235
## 62       2      104       105          3022.043
## 63       2      103       105          3181.984
## 64       2      101       105          4156.250
## 65       2      105       105          3024.107
## 66       2      106       106          4621.252
## 67       2      102       106          2108.811
## 68       2      103       106          2405.058
## 69       2      101       106          3154.517
## 70       2      104       106          2317.747
## 71       2      105       106          3327.778
## 72       3      106       101          5036.317
## 73       3      104       101          4623.781
## 74       3      103       101          5033.457
## 75       3      101       101          6738.287
## 76       3      105       101          4701.737
## 77       3      102       101          5459.661
## 78       3      102       102          3956.177
## 79       3      106       102          4358.268
## 80       3      103       102          3714.736
## 81       3      105       102          3697.386
## 82       3      104       102          3749.649
## 83       3      101       102          4041.438
## 84       3      103       103          3774.709
## 85       3      104       103          3693.235
## 86       3      101       103          3633.333
## 87       3      105       103          3614.320
## 88       3      102       103          3381.650
## 89       3      106       103          4837.681
## 90       3      104       104          3824.913
## 91       3      106       104          5600.000
## 92       3      103       104          3828.069
## 93       3      102       104          3532.500
## 94       3      101       104          3286.910
## 95       3      105       104          4567.164
## 96       3      106       105          5119.863
## 97       3      102       105          3886.532
## 98       3      105       105          3677.961
## 99       3      101       105          3622.430
## 100      3      103       105          3663.843
## 101      3      104       105          3573.110
## 102      3      102       106          2746.346
## 103      3      101       106          3308.428
## 104      3      103       106          2860.274
```

```
## 105       3       104       106          3498.678
## 106       3       105       106          5214.347
## 107       3       106       106          6557.092
## 108       4       106       101          7280.000
## 109       4       103       101          5284.559
## 110       4       104       101          4667.426
## 111       4       102       101          5287.874
## 112       4       105       101          5640.000
## 113       4       101       101          6243.729
## 114       4       102       102          3961.977
## 115       4       101       102          4339.203
## 116       4       106       102          8490.000
## 117       4       105       102          6581.944
## 118       4       104       102          4305.623
## 119       4       103       102          4002.327
## 120       4       106       103          5616.667
## 121       4       101       103          3832.847
## 122       4       102       103          4054.839
## 123       4       103       103          3868.952
## 124       4       104       103          4340.000
## 125       4       105       103          7344.444
## 126       4       101       104          3820.319
## 127       4       106       104          8225.000
## 128       4       105       104          5094.118
## 129       4       102       104          3358.287
## 130       4       104       104          4543.000
## 131       4       103       104          4366.549
## 132       4       105       105          5497.191
## 133       4       106       105          4690.000
## 134       4       104       105          4429.146
## 135       4       103       105          4098.904
## 136       4       101       105          4895.536
## 137       4       102       105          3719.399
## 138       4       105       106          6087.671
## 139       4       101       106          3924.939
## 140       4       103       106          3696.689
## 141       4       104       106          5461.722
## 142       4       102       106          4084.459
## 143       4       106       106          6652.500
## 144       5       102       101          5422.951
## 145       5       103       101          5922.892
## 146       5       106       101          4527.273
## 147       5       101       101          6610.062
## 148       5       105       101          5533.333
## 149       5       104       101          5551.205
## 150       5       105       102          2100.000
## 151       5       104       102          4244.000
## 152       5       102       102          4473.095
## 153       5       106       102          2775.000
## 154       5       103       102          4533.486
```

```
## 155       5      101       102          5162.818
## 156       5      103       103          3928.947
## 157       5      102       103          4918.000
## 158       5      106       103          7525.000
## 159       5      104       103          6350.000
## 160       5      101       103          4843.261
## 161       5      105       103          4858.333
## 162       5      101       104          4379.037
## 163       5      103       104          5010.870
## 164       5      104       104          6376.087
## 165       5      106       104         16000.000
## 166       5      102       104          3715.517
## 167       5      105       104         10733.333
## 168       5      103       105          5148.696
## 169       5      104       105          5008.642
## 170       5      106       105          4812.500
## 171       5      105       105          7018.182
## 172       5      102       105          4544.444
## 173       5      101       105          5516.667
## 174       5      105       106          6731.818
## 175       5      101       106          5618.519
## 176       5      104       106          7263.115
## 177       5      103       106          5690.323
## 178       5      102       106          5495.000
## 179       5      106       106          1200.000
## 180       6      104       101          5780.000
## 181       6      106       101          1200.000
## 182       6      102       101          5909.091
## 183       6      101       101          6557.522
## 184       6      103       101          5916.667
## 185       6      104       102          5634.615
## 186       6      101       102          5843.878
## 187       6      102       102          4334.375
## 188       6      103       102          4381.250
## 189       6      105       102          2850.000
## 190       6      102       103          3400.000
## 191       6      104       103          7483.333
## 192       6      101       103          4839.474
## 193       6      103       103          3894.444
## 194       6      103       104          4855.000
## 195       6      106       104           450.000
## 196       6      104       104          1625.000
## 197       6      102       104          3810.000
## 198       6      101       104          4323.438
## 199       6      102       105          5040.909
## 200       6      106       105          3600.000
## 201       6      103       105          5687.500
## 202       6      105       105         10328.571
## 203       6      104       105          7246.667
## 204       6      101       105          5606.818
```

```
## 205       6      101       106          5244.340
## 206       6      103       106          6242.308
## 207       6      102       106          6373.077
## 208       6      104       106          7955.556
```

- After querying the data, we will free the resources and close the connection to MariaDB:

```
dbClearResult(query.2)
```

The implementation of the RMySQL package with MariaDB is almost the same as with the MySQL database, and both databases can be used interchangeably, depending on your preferences. The only difference is the database server type, its libraries, and specific installation requirements dependent on the operating system of a virtual machine.

- We will now try to connect to MariaDB using the dplyr. It generally works very well with most open source databases. It also allows users with minimal SQL knowledge to run SQL queries without writing SQL queries explicitly; however, this functionality is also allowed in dplyr and enables users to perform more complex queryies.
- We have already installed dplyr, therefore we just need to load the package into R in the standard way:

```
library(dplyr)
```

- For the MariaDB connection with R through dplyr, we can use the credentials as below:

```
dpl.conn <- DBI::dbConnect(RMariaDB::MariaDB(),
  host = "localhost",
  port = 3316,
  dbname = "data1",
  user = "ansari",
  password = rstudioapi::askForPassword("Database password")
)
```

- By calling the name of the connection (the dpl.conn object), we can view some information:

```
dpl.conn

## <MariaDBConnection>
##   Host:    localhost
##   Server:
##   Client:
```

- The tbl() function uses the created connection and provides a snapshot of the referenced table:

```
need.data <- tbl(dpl.conn, "need")
need.data

## # Source:   table<need> [?? x 50]
## # Database: mysql  [ansari@localhost:NA/data1]
```

```
##    hh_id region    imd_eng imd_wales gcons2005 gcons…¹ gcons…² gcons…³
gcons…⁴
##    <int> <chr>     <chr>   <chr>     <chr>     <chr>   <chr>   <chr>
<chr>
##  1     1 E12000007 1       ""        "35000"   V       "24500" V
"22000"
##  2     2 E12000002 4       ""        "19000"   V       "14900" V
"16000"
##  3     3 E12000002 4       ""        "22500"   V       "22500" V
"22500"
##  4     4 E12000005 1       ""        "21000"   V       "20500" V
"18000"
##  5     5 E12000003 1       ""        ""        M       ""      M       ""
##  6     6 E12000007 2       ""        ""        O       ""      O       ""
##  7     7 E12000006 3       ""        "12000"   V       "16500" V
"12300"
##  8     8 E12000005 5       ""        "18500"   V       "15500" V
"13900"
##  9     9 E12000007 4       ""        "35000"   V       "40000" V
"35000"
## 10    10 E12000003 2       ""        "28000"   V       "26000" V
"24000"
## # … with more rows, 41 more variables: gcons2007valid <chr>, gcons2008
<chr>,
## #   gcons2008valid <chr>, gcons2009 <chr>, gcons2009valid <chr>,
## #   gcons2010 <chr>, gcons2010valid <chr>, gcons2011 <chr>,
## #   gcons2011valid <chr>, gcons2012 <chr>, gcons2012valid <chr>,
## #   econs2005 <chr>, econs2005valid <chr>, econs2006 <chr>,
## #   econs2006valid <chr>, econs2007 <chr>, econs2007valid <chr>,
## #   econs2008 <chr>, econs2008valid <chr>, econs2009 <chr>, …
```

- You can also obtain more detailed output on the structure of the need.data tbl_mysql object by using the generic str() function:

```
str(need.data)
```

```
## List of 2
##  $ src       :List of 2
##   ..$ con  :Formal class 'MariaDBConnection' [package "RMariaDB"] with 7
slots
##   .. .. ..@ ptr                   :<externalptr>
##   .. .. ..@ host                  : chr "localhost"
##   .. .. ..@ db                    : chr "data1"
##   .. .. ..@ load_data_local_infile: logi FALSE
##   .. .. ..@ bigint                : chr "integer64"
##   .. .. ..@ timezone              : chr "UTC"
##   .. .. ..@ timezone_out          : chr "UTC"
##   ..$ disco: NULL
##   ..- attr(*, "class")= chr [1:4] "src_MariaDBConnection" "src_dbi"
"src_sql" "src"
##  $ lazy_query:List of 5
```

```
##   ..$ x         : 'ident' chr "need"
##   ..$ vars      : chr [1:50] "hh_id" "region" "imd_eng" "imd_wales" ...
##   ..$ group_vars: chr(0)
##   ..$ order_vars: NULL
##   ..$ frame     : NULL
##   ..- attr(*, "class")= chr [1:3] "lazy_base_remote_query"
"lazy_base_query" "lazy_query"
##  - attr(*, "class")= chr [1:5] "tbl_MariaDBConnection" "tbl_dbi" "tbl_sql"
"tbl_lazy" ...
```

- We will now run a little bit more advanced SQL query on the NEED data. We will calculate the average electricity consumption for the years 2005 through 2012, grouped by geographical region (region) and property type (prop_type). We will order the results by region and property type.
- The dplyr package requires that all activities are performed in sequence. Therefore, we first need to explicitly set the grouping variables (region and prop_type) for the table:

```
by.regiontype <- group_by(need.data, region, prop_type)
by.regiontype
```

```
## # Source:   table<need> [?? x 50]
## # Database: mysql  [ansari@localhost:NA/data1]
## # Groups:   region, prop_type
##    hh_id region    imd_eng imd_wales gcons2005 gcons…¹ gcons…² gcons…³
gcons…⁴
##    <int> <chr>     <chr>   <chr>     <chr>     <chr>   <chr>   <chr>
<chr>
##  1     1 E12000007 1       ""        "35000"   V       "24500" V
"22000"
##  2     2 E12000002 4       ""        "19000"   V       "14900" V
"16000"
##  3     3 E12000002 4       ""        "22500"   V       "22500" V
"22500"
##  4     4 E12000005 1       ""        "21000"   V       "20500" V
"18000"
##  5     5 E12000003 1       ""        ""        M       ""      M       ""
##  6     6 E12000007 2       ""        ""        O       ""      O       ""
##  7     7 E12000006 3       ""        "12000"   V       "16500" V
"12300"
##  8     8 E12000005 5       ""        "18500"   V       "15500" V
"13900"
##  9     9 E12000007 4       ""        "35000"   V       "40000" V
"35000"
## 10    10 E12000003 2       ""        "28000"   V       "26000" V
"24000"
## # … with more rows, 41 more variables: gcons2007valid <chr>, gcons2008
<chr>,
## #   gcons2008valid <chr>, gcons2009 <chr>, gcons2009valid <chr>,
```

```
## #   gcons2010 <chr>, gcons2010valid <chr>, gcons2011 <chr>,
## #   gcons2011valid <chr>, gcons2012 <chr>, gcons2012valid <chr>,
## #   econs2005 <chr>, econs2005valid <chr>, econs2006 <chr>,
## #   econs2006valid <chr>, econs2007 <chr>, econs2007valid <chr>,
## #   econs2008 <chr>, econs2008valid <chr>, econs2009 <chr>, …
```

- Note that the preceding output contains information about both grouping variables added to the structure of the table. This grouped table has been stored as a new object named by.regiontype. This new grouped object will now be used to calculate the average electricity consumption for each year (from 2005 until 2012) aggregated by both grouping variables defined earlier:

```
avg.elec <- summarise(by.regiontype,
                      elec2005 = mean(econs2005),
                      elec2006 = mean(econs2006),
                      elec2007 = mean(econs2007),
                      elec2008 = mean(econs2008),
                      elec2009 = mean(econs2009),
                      elec2010 = mean(econs2010),
                      elec2011 = mean(econs2011),
                      elec2012 = mean(econs2012))
```

- Finally, we can order the resulting table by region and property type. By default, the arrange() function sorts the values in ascending order:

```
avg.elec <- arrange(avg.elec, region, prop_type)
avg.elec

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

## # Source:     SQL [?? x 10]
## # Database:   mysql  [ansari@localhost:NA/data1]
## # Groups:     region
## # Ordered by: region, prop_type
##    region    prop_type elec2…¹ elec2…² elec2…³ elec2…⁴ elec2…⁵ elec2…⁶
elec2…⁷
##    <chr>         <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
<dbl>
##  1 E12000001       101   5341.   5196.   5299.   4863.   4780.   4947.
4819.
##  2 E12000001       102   3841.   3757.   3733.   3524.   3450.   3493.
3593.
##  3 E12000001       103   3735.   3816.   3891.   3676.   3555.   3479.
3534.
##  4 E12000001       104   3709.   3702.   3617.   3373.   3263.   3185.
3375.
##  5 E12000001       105   3337.   3347.   3278.   3144.   3129.   3017.
3260.
##  6 E12000001       106   3009.   3010.   2954.   2935.   2974.   2865.
2858.
##  7 E12000002       101   5277.   5532.   5415.   5124.   5157.   5059.
```

```
5152.
##  8 E12000002       102   4384.   4347.   4262.   3913.   3876.   3873.
3943.
##  9 E12000002       103   3809.   4140.   3955.   3741.   3659.   3625.
3767.
## 10 E12000002       104   3727.   3716.   3694.   3473.   3365.   3291.
3459.
## # … with more rows, 1 more variable: elec2012 <dbl>, and abbreviated
variable
## #   names ¹elec2005, ²elec2006, ³elec2007, ⁴elec2008, ⁵elec2009, ⁶
elec2010,
## #   ⁷elec2011
```

- This is the final output of the results set of our query. You are probably wondering right now why we use the word query if we didn't even write a single SQL command during our data processing. Well, in fact we did. The dplyr package is extremely user friendly and it doesn't require its users to understand and know Structured Query Language – although this knowledge would undoubtedly help as some of the errors are related to how the SQL queries are constructed. However, it translates its functions, such as group_by(), summarise(), arrange(), and many others, into their SQL equivalents in the background.
- If you are curious, how dplyr does it, you may use show_query() or, even better, the explain() function, which prints the applied SQL query and its plan, for example:

```
show_query(avg.elec)

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

## <SQL>
## SELECT
##   `region`,
##   `prop_type`,
##   AVG(`econs2005`) AS `elec2005`,
##   AVG(`econs2006`) AS `elec2006`,
##   AVG(`econs2007`) AS `elec2007`,
##   AVG(`econs2008`) AS `elec2008`,
##   AVG(`econs2009`) AS `elec2009`,
##   AVG(`econs2010`) AS `elec2010`,
##   AVG(`econs2011`) AS `elec2011`,
##   AVG(`econs2012`) AS `elec2012`
## FROM `need`
## GROUP BY `region`, `prop_type`
## ORDER BY `region`, `prop_type`

explain(avg.elec)

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

```
## <SQL>
## SELECT
##   `region`,
##   `prop_type`,
##   AVG(`econs2005`) AS `elec2005`,
##   AVG(`econs2006`) AS `elec2006`,
##   AVG(`econs2007`) AS `elec2007`,
##   AVG(`econs2008`) AS `elec2008`,
##   AVG(`econs2009`) AS `elec2009`,
##   AVG(`econs2010`) AS `elec2010`,
##   AVG(`econs2011`) AS `elec2011`,
##   AVG(`econs2012`) AS `elec2012`
## FROM `need`
## GROUP BY `region`, `prop_type`
## ORDER BY `region`, `prop_type`
##
## <PLAN>

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

##   id select_type table type possible_keys  key key_len  ref  rows
## 1  1      SIMPLE  need  ALL          <NA> <NA>    <NA> <NA> 49253
##                            Extra
## 1 Using temporary; Using filesort
```

- It's important to stress that all this processing has occurred within the database, without affecting the performance of R. We may now pull the results set from the database into R as a data.frame object. To be precise, as we grouped the table by two grouping variables, our new object is now a grouped data frame (grouped_df):

```
elec.df <- collect(avg.elec)

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

elec.df

## # A tibble: 60 × 10
## # Groups:   region [10]
##    region    prop_type elec2…¹ elec2…² elec2…³ elec2…⁴ elec2…⁵ elec2…⁶
elec2…⁷
##    <chr>         <int>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
<dbl>
##  1 E12000001       101   5341.   5196.   5299.   4863.   4780.   4947.
4819.
##  2 E12000001       102   3841.   3757.   3733.   3524.   3450.   3493.
3593.
##  3 E12000001       103   3735.   3816.   3891.   3676.   3555.   3479.
3534.
##  4 E12000001       104   3709.   3702.   3617.   3373.   3263.   3185.
3375.
```

```
##  5 E12000001       105   3337.   3347.   3278.   3144.   3129.   3017.
3260.
##  6 E12000001       106   3009.   3010.   2954.   2935.   2974.   2865.
2858.
##  7 E12000002       101   5277.   5532.   5415.   5124.   5157.   5059.
5152.
##  8 E12000002       102   4384.   4347.   4262.   3913.   3876.   3873.
3943.
##  9 E12000002       103   3809.   4140.   3955.   3741.   3659.   3625.
3767.
## 10 E12000002       104   3727.   3716.   3694.   3473.   3365.   3291.
3459.
## # … with 50 more rows, 1 more variable: elec2012 <dbl>, and abbreviated
## #   variable names ¹elec2005, ²elec2006, ³elec2007, ⁴elec2008, ⁵elec2009,
## #   ⁶elec2010, ⁷elec2011
```

In order to convert the grouped_df into an R native data.frame, we can simply use as.data.frame() from base R:

```
elec <- as.data.frame(elec.df)
elec
```

```
##       region prop_type elec2005 elec2006 elec2007 elec2008 elec2009
elec2010
## 1  E12000001       101 5341.386 5196.255 5298.689 4862.547 4779.775
4947.191
## 2  E12000001       102 3840.788 3757.433 3733.164 3523.888 3450.191
3493.393
## 3  E12000001       103 3734.703 3816.210 3890.868 3676.256 3554.566
3478.539
## 4  E12000001       104 3709.131 3701.773 3617.465 3372.784 3262.766
3185.284
## 5  E12000001       105 3337.374 3346.970 3278.114 3144.276 3128.620
3017.340
## 6  E12000001       106 3009.375 3010.417 2954.167 2934.635 2974.479
2865.495
## 7  E12000002       101 5276.891 5531.513 5415.006 5123.770 5157.143
5058.884
## 8  E12000002       102 4384.243 4346.923 4261.663 3912.655 3876.253
3873.002
## 9  E12000002       103 3809.194 4140.323 3954.597 3741.290 3658.548
3625.242
## 10 E12000002       104 3726.642 3715.623 3693.892 3473.204 3364.764
3291.191
## 11 E12000002       105 3930.464 3823.593 3879.139 3585.844 3434.685
3550.414
## 12 E12000002       106 3349.447 3287.389 3397.345 3244.414 3176.272
3219.580
## 13 E12000003       101 5625.307 5665.031 5606.288 5243.712 5245.475
5078.298
```

```
## 14 E12000003       102 4266.611 4045.126 3986.505 3577.288 3588.727
3624.436
## 15 E12000003       103 3754.931 3849.197 3746.560 3678.670 3588.532
3548.739
## 16 E12000003       104 3803.040 3678.765 3667.072 3484.752 3393.732
3163.237
## 17 E12000003       105 4088.529 3975.044 3826.357 3468.914 3388.792
3395.884
## 18 E12000003       106 3280.837 3253.704 3330.274 3333.092 3384.863
3270.612
## 19 E12000004       101 5467.987 5493.344 5526.843 5189.824 5028.163
5010.066
## 20 E12000004       102 4276.227 4249.673 4169.599 4017.921 3779.378
3732.610
## 21 E12000004       103 4222.424 4144.848 4079.697 4092.273 3759.242
3796.212
## 22 E12000004       104 3778.049 3711.204 3654.802 3749.238 3621.341
3580.945
## 23 E12000004       105 4004.375 4030.714 4111.607 3901.518 3745.446
3717.143
## 24 E12000004       106 3203.444 3185.842 3349.617 3224.362 3142.857
3077.041
## 25 E12000005       101 5282.525 5470.349 5299.887 5122.379 5136.020
5064.318
## 26 E12000005       102 4416.143 4460.835 4347.644 4028.141 3859.579
3822.111
## 27 E12000005       103 4094.869 4280.459 4492.358 3942.140 3835.590
3833.406
## 28 E12000005       104 3836.102 3938.095 3749.779 3604.983 3564.950
3520.210
## 29 E12000005       105 3961.041 4197.462 4075.635 3799.112 3744.036
3559.645
## 30 E12000005       106 3824.222 3875.481 3822.815 3715.481 3668.222
3637.111
## 31 E12000006       101 5642.992 5803.434 5902.775 5573.754 5484.995
5441.957
## 32 E12000006       102 4779.944 4731.437 4655.287 4398.063 4186.400
4290.839
## 33 E12000006       103 4151.174 4490.509 4366.928 4116.634 3938.356
4077.299
## 34 E12000006       104 4080.721 4044.222 4002.460 3816.247 3774.085
3783.238
## 35 E12000006       105 4642.944 4635.685 4499.261 4225.941 4089.113
4029.032
## 36 E12000006       106 3573.580 3579.136 3635.741 3619.691 3502.469
3553.086
## 37 E12000007       101 6526.923 6600.366 6621.429 6404.579 6293.223
6433.333
## 38 E12000007       102 4883.366 4922.049 4921.854 4599.805 4534.976
4497.854
```

```
## 39 E12000007       103 4631.688 4612.478 4529.892 4444.345 4490.575
4103.770
## 40 E12000007       104 4602.691 4401.381 4422.486 4304.993 4178.506
4196.884
## 41 E12000007       105 4547.984 4365.323 4554.032 4036.290 4006.048
4082.661
## 42 E12000007       106 3535.260 3459.140 3472.497 3397.167 3314.229
3306.225
## 43 E12000008       101 6209.545 6081.347 6078.634 5832.782 5798.097
5855.427
## 44 E12000008       102 4754.385 4726.110 4756.748 4401.765 4343.508
4261.048
## 45 E12000008       103 4217.486 4278.613 4275.434 4145.809 3990.679
3837.789
## 46 E12000008       104 4131.225 4079.222 4008.720 3886.881 3791.005
3814.375
## 47 E12000008       105 4570.487 4315.914 4306.651 4250.891 4143.943
4239.549
## 48 E12000008       106 3540.457 3558.112 3721.404 3653.613 3552.147
3443.354
## 49 E12000009       101 5511.465 5543.896 5569.586 5350.000 5208.227
5207.909
## 50 E12000009       102 4746.602 4580.946 4565.060 4326.079 4263.912
4161.478
## 51 E12000009       103 4564.238 4371.965 4367.219 4050.000 4158.389
4049.338
## 52 E12000009       104 4176.600 4166.410 4028.318 3748.104 3660.664
3743.187
## 53 E12000009       105 4860.610 4603.924 4456.904 4548.910 4456.686
4246.439
## 54 E12000009       106 3878.815 3753.909 3749.622 3830.769 3694.325
3641.803
## 55 W99999999       101 5582.536 5328.067 5249.480 5295.530 5199.376
5209.771
## 56 W99999999       102 4214.477 4082.507 3889.209 3819.236 3722.185
3681.568
## 57 W99999999       103 4112.795 3981.890 3918.110 3901.181 3780.315
3589.173
## 58 W99999999       104 3930.964 3742.970 3530.411 3355.687 3289.415
3375.987
## 59 W99999999       105 4501.360 4476.435 4047.130 4109.517 3898.187
3906.344
## 60 W99999999       106 2853.311 2803.974 2841.722 2602.318 2782.450
2767.550
##    elec2011 elec2012
## 1  4819.476 4893.071
## 2  3593.329 3549.047
## 3  3534.247 3508.676
## 4  3375.443 3251.862
## 5  3259.596 3265.488
```

```
## 6  2858.203 2913.411
## 7  5151.501 4954.622
## 8  3943.077 3859.677
## 9  3766.774 3702.823
## 10 3459.484 3388.183
## 11 3701.738 3675.248
## 12 3313.717 3236.947
## 13 5205.598 5084.126
## 14 3712.666 3622.381
## 15 3608.716 3597.592
## 16 3286.950 3255.519
## 17 3393.958 3349.475
## 18 3235.185 3331.562
## 19 5071.122 4869.912
## 20 3818.494 3773.118
## 21 3798.182 3673.333
## 22 3583.155 3398.095
## 23 3615.089 3819.196
## 24 3173.214 3349.107
## 25 5078.410 4838.275
## 26 3873.492 3852.167
## 27 3835.044 3793.886
## 28 3524.474 3568.992
## 29 3769.924 3603.173
## 30 3658.593 3648.519
## 31 5534.102 5436.077
## 32 4233.656 4166.586
## 33 3946.869 3889.530
## 34 3769.622 3749.027
## 35 4135.618 4233.468
## 36 3417.160 3367.284
## 37 6843.407 6616.300
## 38 4693.610 4730.341
## 39 4237.792 4172.621
## 40 4190.227 4077.939
## 41 3940.726 4046.774
## 42 3328.228 3356.686
## 43 5697.910 5670.025
## 44 4239.208 4118.821
## 45 3873.410 3794.075
## 46 3861.587 3777.141
## 47 4367.280 4276.188
## 48 3586.469 3542.399
## 49 5324.257 5210.403
## 50 4130.808 4183.838
## 51 3931.126 3844.260
## 52 3732.227 3726.718
## 53 4326.163 4312.936
## 54 3581.463 3613.997
## 55 5189.709 5124.012
```

```
## 56 3745.845 3753.083
## 57 3808.465 3738.976
## 58 3526.540 3526.856
## 59 4038.369 3877.492
## 60 2887.748 2725.828
```

- Of course, a large amount of aggregated electricity consumption is not too meaningful in this format. If you want to present this data graphically, you should probably transform the data from wide into narrow/long format. You can achieve this through the reshape() function:

```
elec.l <- reshape(elec,
                  varying = c("elec2005", "elec2006", "elec2007",
                              "elec2008", "elec2009", "elec2010",
                              "elec2011", "elec2012"),
                  v.names = "electricity",
                  timevar = "year",
                  times = c("2005", "2006", "2007", "2008",
                            "2009", "2010", "2011", "2012"),
                  direction = "long")
```

- The varying parameter refers to variables that you want to reshape from wide into long format. Putting it simply, the variables/columns specified in the varying option will be converted into a single variable named in the timevar parameter.
- The labels of varying variables will be used as categorical values of the new variable specified in timevar, but you can re-label them in the times parameter.
- The v.names parameter sets the name for the new variable, which will take over the values of average electricity consumption for each year.
- The resulting data.frame will look as follows:

```
head(elec.l, n=25)
```

```
##            region prop_type year electricity id
## 1.2005  E12000001       101 2005    5341.386  1
## 2.2005  E12000001       102 2005    3840.788  2
## 3.2005  E12000001       103 2005    3734.703  3
## 4.2005  E12000001       104 2005    3709.131  4
## 5.2005  E12000001       105 2005    3337.374  5
## 6.2005  E12000001       106 2005    3009.375  6
## 7.2005  E12000002       101 2005    5276.891  7
## 8.2005  E12000002       102 2005    4384.243  8
## 9.2005  E12000002       103 2005    3809.194  9
## 10.2005 E12000002       104 2005    3726.642 10
## 11.2005 E12000002       105 2005    3930.464 11
## 12.2005 E12000002       106 2005    3349.447 12
## 13.2005 E12000003       101 2005    5625.307 13
## 14.2005 E12000003       102 2005    4266.611 14
## 15.2005 E12000003       103 2005    3754.931 15
## 16.2005 E12000003       104 2005    3803.040 16
## 17.2005 E12000003       105 2005    4088.529 17
## 18.2005 E12000003       106 2005    3280.837 18
```

```
## 19.2005 E12000004       101 2005    5467.987 19
## 20.2005 E12000004       102 2005    4276.227 20
## 21.2005 E12000004       103 2005    4222.424 21
## 22.2005 E12000004       104 2005    3778.049 22
## 23.2005 E12000004       105 2005    4004.375 23
## 24.2005 E12000004       106 2005    3203.444 24
## 25.2005 E12000005       101 2005    5282.525 25
```

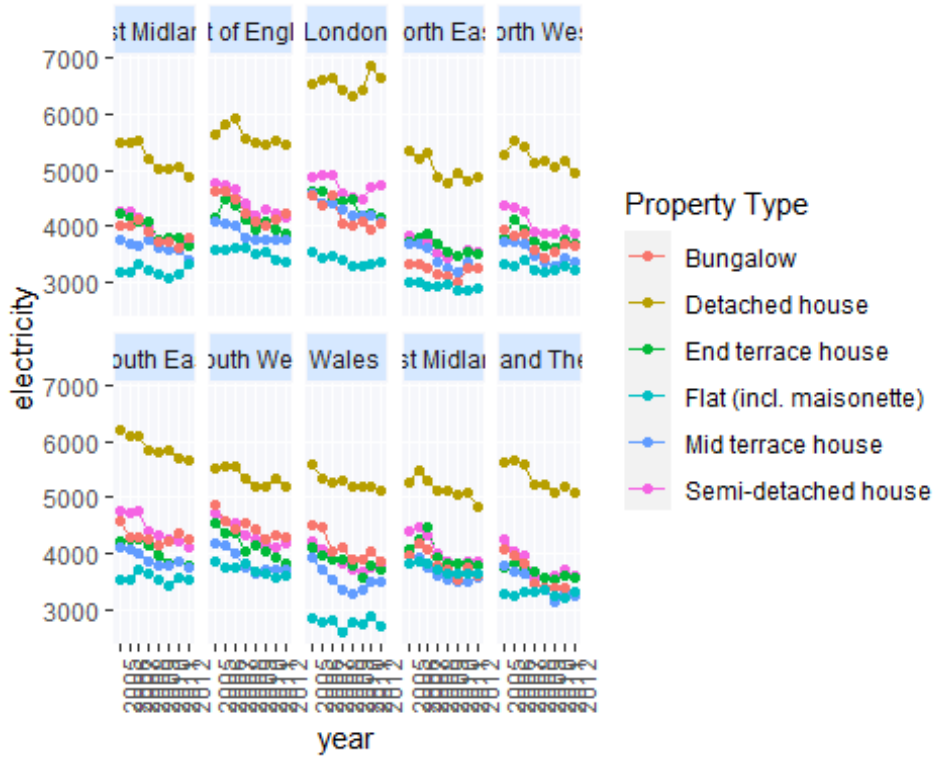
- We can also do some extra tidying up of the labels for the region and prop_type variables to make them more human readable and user-friendly.
- For that reason, we will re-label all values for these two variables according to the data dictionary file for the NEED dataset, available at https://www.gov.uk/government/statistics/national-energy-efficiency-d ata-framework-need-anonymised-data-2014 (see the Look up tables Excel spreadsheet file for details):

```
elec.l <- within(elec.l, {
  region[region=="E12000001"] <- "North East"
  region[region=="E12000002"] <- "North West"
  region[region=="E12000003"] <- "Yorkshire and The Humber"
  region[region=="E12000004"] <- "East Midlands"
  region[region=="E12000005"] <- "West Midlands"
  region[region=="E12000006"] <- "East of England"
  region[region=="E12000007"] <- "London"
  region[region=="E12000008"] <- "South East"
  region[region=="E12000009"] <- "South West"
  region[region=="W99999999"] <- "Wales"
  })

elec.l <- within(elec.l, {
  prop_type[prop_type==101] <- "Detached house"
  prop_type[prop_type==102] <- "Semi-detached house"
  prop_type[prop_type==103] <- "End terrace house"
  prop_type[prop_type==104] <- "Mid terrace house"
  prop_type[prop_type==105] <- "Bungalow"
  prop_type[prop_type==106] <- "Flat (incl. maisonette)"
  })

head(elec.l, n=6)
```

```
##            region               prop_type year electricity id
## 1.2005 North East          Detached house 2005    5341.386  1
## 2.2005 North East     Semi-detached house 2005    3840.788  2
## 3.2005 North East       End terrace house 2005    3734.703  3
## 4.2005 North East       Mid terrace house 2005    3709.131  4
## 5.2005 North East                 Bungalow 2005    3337.374  5
## 6.2005 North East Flat (incl. maisonette) 2005    3009.375  6
```

- Finally, using the ggplot2 package, we can visualize the obtained results in a more informative way:

```
library(ggplot2)
ggplot(elec.l,
       aes(x=year, y=electricity,
           group=factor(prop_type),
           colour=factor(prop_type))) +
  geom_line() + geom_point() +
  facet_wrap(~region, nrow = 2) +
  scale_colour_discrete(name="Property Type") +
  theme(axis.text.x = element_text(angle = 90),
        panel.grid.major=element_line(colour = "white"),
        panel.grid.minor=element_blank(),
        panel.background=element_rect(fill = "#f6f7fb"),
        strip.background = element_rect(colour = "#f6f7fb",
                                        fill = "#d6e8ff"))
```



The preceding code produces multiple line plots of the average electricity consumption by property type across all years of NEED data, presented for each level of the geographical region variable (facet_wrap(~region, nrow = 2)).

- We can clearly observe some patterns in the data, for example that detached houses have been consuming on average much more electricity than other property types across several years, but this doesn't surprise us considering that detached houses usually contain more rooms, are poorly insulated, and generally accommodate more residents.
- It is, however, striking that in general, most households show a tendency to lower their electricity consumption from year to year.

- We can only speculate as to what the reason for this behavior might be. Do people become more environmentally friendly? Are we too busy to stay indoors for a long time? Are our houses better insulated? Or maybe we simply use less electricity due to increasing energy prices?

```
dbDisconnect(conn)
```