Stat 405/705
Class 3
Statistical computing with R

Richard P. Waterman

Wharton

Table of contents I

Today's module

Topics to be covered in this module:

- Last time
- Data frames
- Reading data from various sources
- Analytics engine architecture
- What you should be able to do now
- Next time

Last time

- Data structures
  1. Vectors
  2. Matrices
  3. Arrays
  4. Lists
- Extraction
- Replacement

Dataframes

- The natural *structured data* container is a matrix.
- For example, the familiar Excel spreadsheet or the JMP data table.
- In R, the matrix structure will only accept elements of the same type.
- A data frame, is a cross between a matrix and a list: it has a rows/columns structure, but its columns may be of different types.
- This makes it the ideal structure for data analysis.
- All the extraction and replacement operations we saw on matrices will work on data frames too.
- The key command to make one is `data.frame`

Example

We will create three vectors: one numeric, one character and the other logical and put them in a data frame.

```
my.vec1 <- c(5.2, 7.6, 15.1,12)  # A vector of  numerics
my.vec2 <- c("Male","Male","Female","Female") # A vector of  characters
my.vec3 <- c(TRUE, FALSE,TRUE,FALSE) # A vector of logicals

my.df       <- data.frame(
                 Numbers = my.vec1,
                 Gender = my.vec2,
                 Logicals = my.vec3
                 )
summary(my.df) #The summary function describes each variable
```

```
##     Numbers          Gender   Logicals
##  Min.   : 5.200   Female:2   Mode :logical
##  1st Qu.: 7.000   Male  :2   FALSE:2
##  Median : 9.800              TRUE :2
##  Mean   : 9.975
##  3rd Qu.:12.775
##  Max.   :15.100
```

Coercion in data frames

Because a data frame is specialized for data analysis and modelling, it will (by default) turn character vectors into factors automatically.

```
#### Extract the Gender column and look at its type
class(my.df$Gender)

## [1] "factor"
```

```
#### You can coerce a factor to a character vector if you want.
new.gender <- as.character(my.df$Gender)
class(new.gender)

## [1] "character"

#### This can be important because some functions only work on characters.
```

Extraction

It's easy to pull out columns and rows from a data frame using the same syntax as for matrices:

```
#Extract the 2nd & 3rd columns, 1st & 2nd rows
my.df[c(1,2),c(2,3)] # A piece of the data frame

##   Gender Logicals
## 1   Male     TRUE
## 2   Male    FALSE

#The dim command tells you the size of the data frame
dim(my.df)

## [1] 4 3
```

Adding rows and columns to a data frame

The commands *cbind* (column bind) and *rbind* (rows bind) let you add rows and columns to an existing data frame. They also work on matrices.

```
my.df <- cbind(my.df,"New column" = c(1,2,3,4)) #Add a column
my.df <- rbind(my.df, c(1,"Male",FALSE,1.1))    # Add a row
print(my.df)

##   Numbers Gender Logicals New column
## 1     5.2   Male     TRUE          1
## 2     7.6   Male    FALSE          2
## 3    15.1 Female     TRUE          3
## 4      12 Female    FALSE          4
## 5       1   Male    FALSE        1.1

#### Be careful. For factors you can't directly add new levels.
#### The levels must already exist.

my.df <- rbind(my.df, c(1,"No answer",FALSE,1.3))

## Warning in `[<-.factor`(`*tmp*`, ri, value = "No answer"):  invalid
factor level, NA generated
```

Overwriting existing entries in a data frame

Just as with vectors and matrices you can overwrite a particular entry:

```
my.df[6,"Gender"] <- "Male"
print(my.df)
```

```
##   Numbers Gender Logicals New column
## 1     5.2   Male     TRUE          1
## 2     7.6   Male    FALSE          2
## 3    15.1 Female     TRUE          3
## 4      12 Female    FALSE          4
## 5       1   Male    FALSE        1.1
## 6       1   Male    FALSE        1.3
```

Reading data into R

There are a number of ways of reading data into R:

- The most common way is from a file resident on the same machine as R is running.
- You can also read from a file on a remote source, via the `http` protocol, just like a web page.
- You can set up a direct link to a SQL database.
- Once you have your data read in, you may wish to save it as a native R data file, with extension ".Rdata".
- You save an Rdata file with the `save command` and read it back in with the `load` command.
- The .Rdata representation should be robust across operating systems.

Reading data from a file

- The most general file reading command is `scan`, but it is not that user friendly.
- The commands `read.table` and `read.csv` are much more user friendly.
- They both read data straight into a data frame and coerce character to factor variables.
- My suggestion is to request data as a CSV file whenever possible. This will make your life much easier when it comes to reading it in to R.

Example, of reading a local file

Reading a local file into R with `read.csv"`:

```
#You need to substitute your own directory path here
#Note the double \\. An alternative is to use "/" in the path
my.data <- read.csv(
file =
 "C:\\Users\\richardw\\Dropbox (Penn)\\Teaching\\705s2019\\Data\\Pharma_EV.
        )
head(my.data,5) #The head command lets you see the top of a file.
```

```
##           DRUG   EV  Prob
## 1       Athsat 1313 0.900
## 2   Bittamucin 1292 0.800
## 3       Catana 9343 0.050
## 4 Comanapracil 2343 0.125
## 5      Dioxnyl 4231 0.100
```

Example, of reading a remote file

Data from market research at a Gala event 

```
# Note the http prefix indicating a remote source
gala.data <- read.csv(file="http://mathmba.com/richardw/gala.csv")
head(gala.data,n=3)

##   RespondentID     IP.Address
## 1   4494303323  70.215.86.xxx
## 2   4479607312 71.230.246.xxx
## 3   4478255195   74.109.9.xxx
##   On.a.scale.of.1.to.5..please.rate.your.overall.opinion.of.the.Black.Ti
## 1
## 2
## 3
##   What.did.you.like..span.style..text.decoration..underline...best..span
## 1
## 2
## 3

# It would be a good idea to fix up the column names
# with the colnames command
```

The `table` and *sort* commands

Let's have a look at all the IP addresses: The command `table` creates a table of the vector offered to it (or a crosstabs, if there is more than one variable).

```
ip.table <- table(gala.data$IP.Address) #Create the table
print(ip.table[1:10]) # Look at just the first 10 entries

##
##  100.14.68.xxx 100.34.135.xxx 100.34.139.xxx 100.34.155.xxx
##              1              1              1              1
##  100.34.45.xxx  100.34.51.xxx 104.254.17.xxx 108.16.231.xxx
##              1              1              2              1
##  108.2.168.xxx  108.2.195.xxx
##              1              1
```

Sorting the table

It would be helpful to sort the table to get the top IP frequencies. 

```
# The sort command has an argument, "decreasing" that controls the order
# We sort first, then pull off the top 15
sort(ip.table,decreasing = TRUE)[1:15]
```

```
##
##  70.215.86.xxx 73.150.163.xxx  70.215.87.xxx 104.254.17.xxx
##              5               4              3               2
##  108.2.212.xxx  108.24.91.xxx 166.171.57.xxx 69.249.105.xxx
##              2               2              2               2
##    70.91.2.xxx 71.225.253.xxx   73.141.9.xxx  75.88.141.xxx
##              2               2              2               2
##  76.116.44.xxx   76.99.53.xxx  100.14.68.xxx
##              2               2              1
```

Maybe some people are taking the survey multiple times?

Working on character strings

- We often want to break up a character string into parts (break the IP address into "subnets" or turn a sentence into individual words.
- The `strsplit` command will break up a string according to a breaking character (which can be a space). 
- We will break up the IP address by the period character.

Working on character strings

Breaking up a character string:

```
#Special characters may need to be escaped \\. for .
strsplit(gala.data[,"IP.Address"],"\\.")

## Error in strsplit(gala.data[, "IP.Address"], "\\."):  non-character
argument

# But strsplit will only work on characters, not factors
# This works and returns a list (I'm just getting the first 3 elements)
strsplit(as.character(gala.data[,"IP.Address"]),"\\.")[1:3]

## [[1]]
## [1] "70"  "215" "86"  "xxx"
##
## [[2]]
## [1] "71"  "230" "246" "xxx"
##
## [[3]]
## [1] "74"  "109" "9"   "xxx"
```

Turning a list into a vector

If you want to turn a list into a vector, `unlist` is the command. When using variables names, R is case sensitive: `IP.address != IP.Address`.

```
# Notice how you can chain the functions together.
# The output of one command becomes the input to the next.
unlist(strsplit(as.character(gala.data[,"IP.Address" ]),"\\."))
```

```
##   [1] "70"  "215" "86"  "xxx" "71"  "230" "246" "xxx" "74"
##  [10] "109" "9"   "xxx" "71"  "185" "210" "xxx" "50"  "29"
##  [19] "216" "xxx" "71"  "224" "193" "xxx" "73"  "13"  "112"
##  [28] "xxx" "73"  "195" "71"  "xxx" "76"  "99"  "53"  "xxx"
##  [37] "73"  "141" "157" "xxx" "100" "34"  "135" "xxx" "76"
##  [46] "116" "44"  "xxx" "71"  "207" "114" "xxx" "108" "2"
##  [55] "168" "xxx" "50"  "191" "214" "xxx" "108" "2"   "212"
##  [64] "xxx" "144" "160" "98"  "xxx" "69"  "242" "0"   "xxx"
##  [73] "209" "203" "79"  "xxx" "71"  "224" "153" "xxx" "108"
##  [82] "16"  "231" "xxx" "70"  "215" "86"  "xxx" "74"  "103"
##  [91] "174" "xxx" "108" "24"  "91"  "xxx" "68"  "84"  "32"
## [100] "xxx" "73"  "150" "163" "xxx" "73"  "150" "163" "xxx"
## [109] "70"  "215" "69"  "xxx" "159" "14"  "243" "xxx" "73"
```

In class activity. Gala event

The director of a Auto Show Gala event has called you. She wants preliminary results regarding what the attendees liked most about the event. In addition, were there any cars that seemed especially popular?

There is a data dump of the survey on-line at:
"http://mathmba.com/richardw/gala.csv" which contains updated results from some ongoing market research.
Pull the data and answer the question.

In class activity. Gala event

Steps:

1. Create a new project for the activity.
2. Download the data.
3. Review and find the appropriate column containing what the attendee liked most about the event.
4. Extract this column from the data frame and break the responses into individual words, breaking on the space character.
5. Create and sort the table of word frequencies.
6. Manually review the most common words to answer the question.


The command `toupper` will make a character string all upper case.

In class activity: discussion

Reading from a database

WARNING! Cutting and pasting the subsequent code will not work for you. Database drivers must be first installed and the firewall reconfigured.

- More complex than reading from a file, but it puts R into an analytics architecture workflow
- Step 1. Install the appropriate database driver for your machine. My database is MySQL and the drivers are at "https://dev.mysql.com/downloads/connector/odbc/".
- Step 2. Configure the connection to talk to the database. (Requires a username and a password, see last slide in today's notes). If you want to make this happen, let me know your IP address.
- Step 3. Add the `RODBC` library to R.
- Step 4. Within R, make the connection to the database.
- Step 5. Write your SQL queries in R and submit to the database.
- Step 6. Close the database connection in R .

I have added a document (I am not the author) to the Misc directory on Canvas describing these steps. It is called `database-connect.pdf`.

Reading from a database

Assume steps 1 and 2 have been completed.

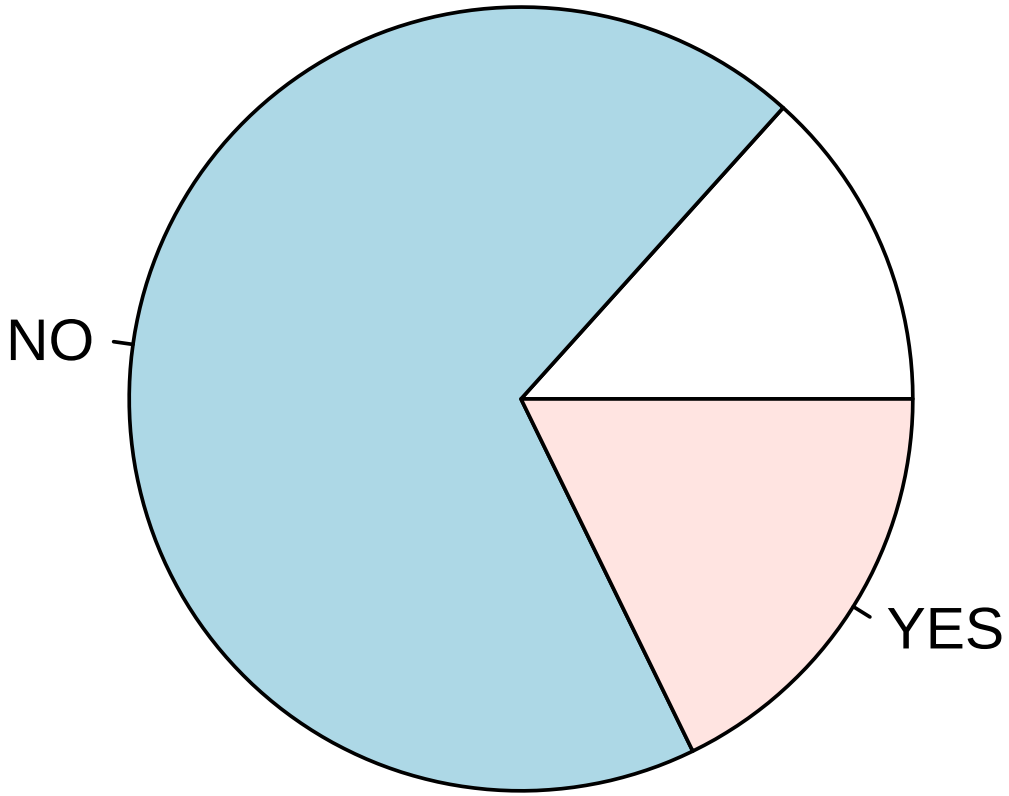
```
#Step 3. #Read in the database connection library
library(RODBC)
#Step 4. connect to database
my.channel <- odbcConnect("STAT705X", uid="stat705_student",
                                        pwd="$[hOCC*TtKO~");
#Step 5. Run the SQL query using the SELECT SQL command to get data
query.result <- sqlQuery(channel = my.channel,
                  query = "SELECT * FROM RESPONSES")
head(query.result,3) # A look at what's in the result data frame

##       MYID Q1   Q2  Q3
## 1  4130869  B WEST YES
## 2 94425955 A- WEST  NO
## 3  5555777

close(my.channel);
```

Reading from a database

```
pie(table(query.result$Q3)) # Just for fun, make a pie chart
```

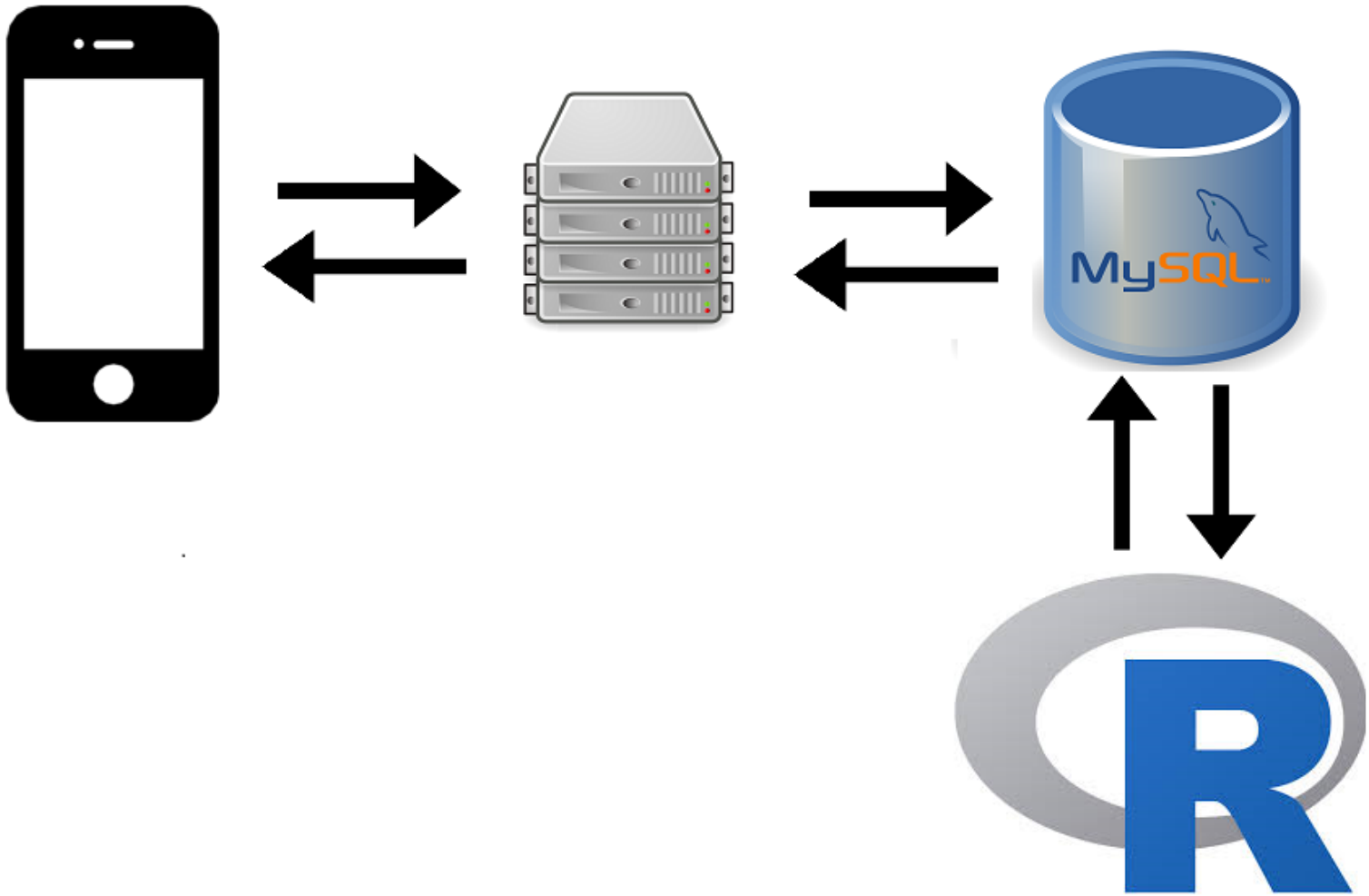


Analytics pipeline architecture: level 1

Just R and a database. The lonely (but productive) analyst.

Analytics pipeline architecture: level 2

R as the off-line analytics engine in a client-server architecture

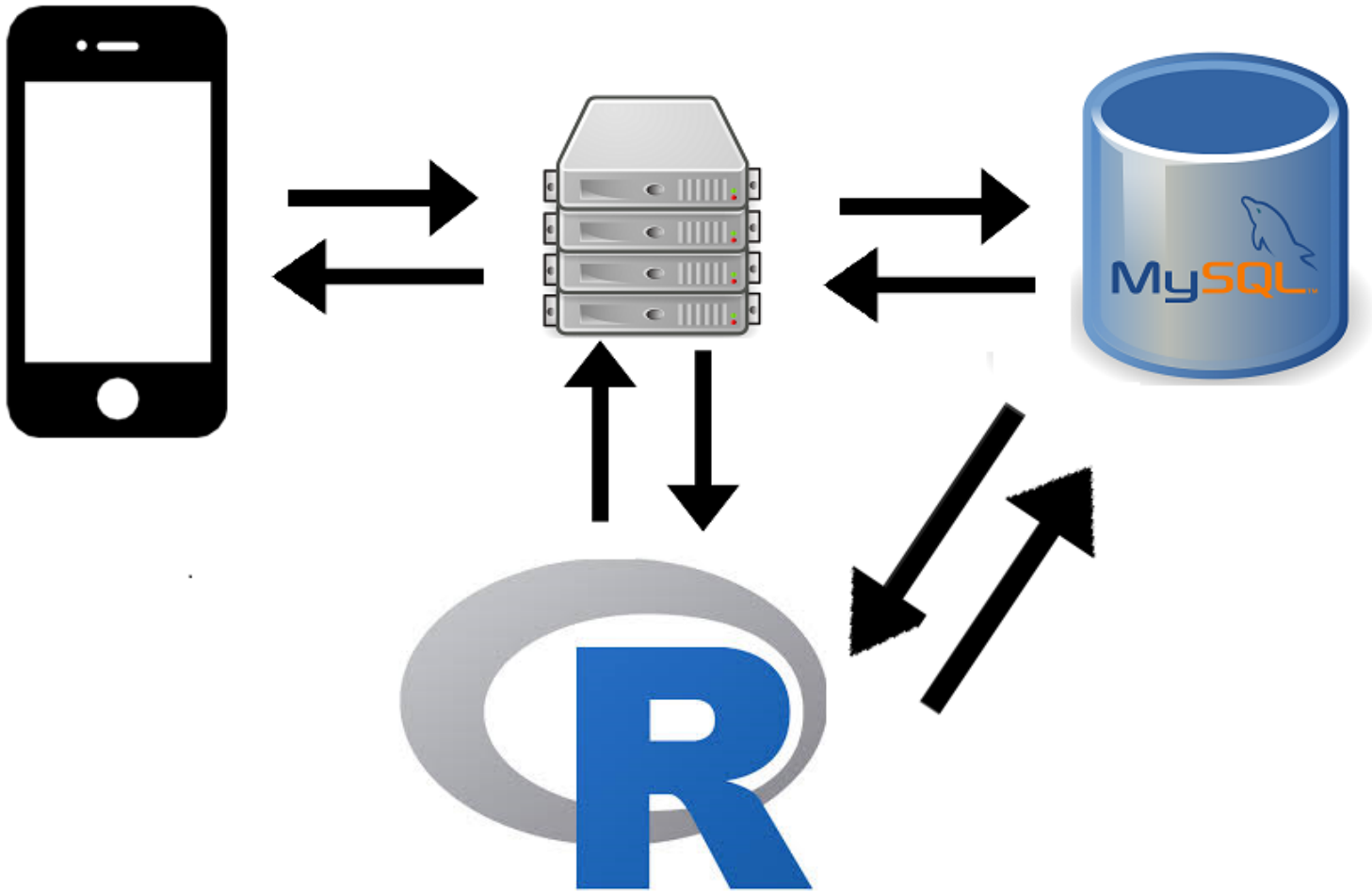


Analytics pipeline architecture: level 2

- Check out: http://mathmba.com/question.html
- New data is entered by the client and the subsequent data pull from R has access to the updated data immediately.

Analytics pipeline architecture: level 3

R as the on-line analytics engine in a client-server architecture (it is being called by the server)

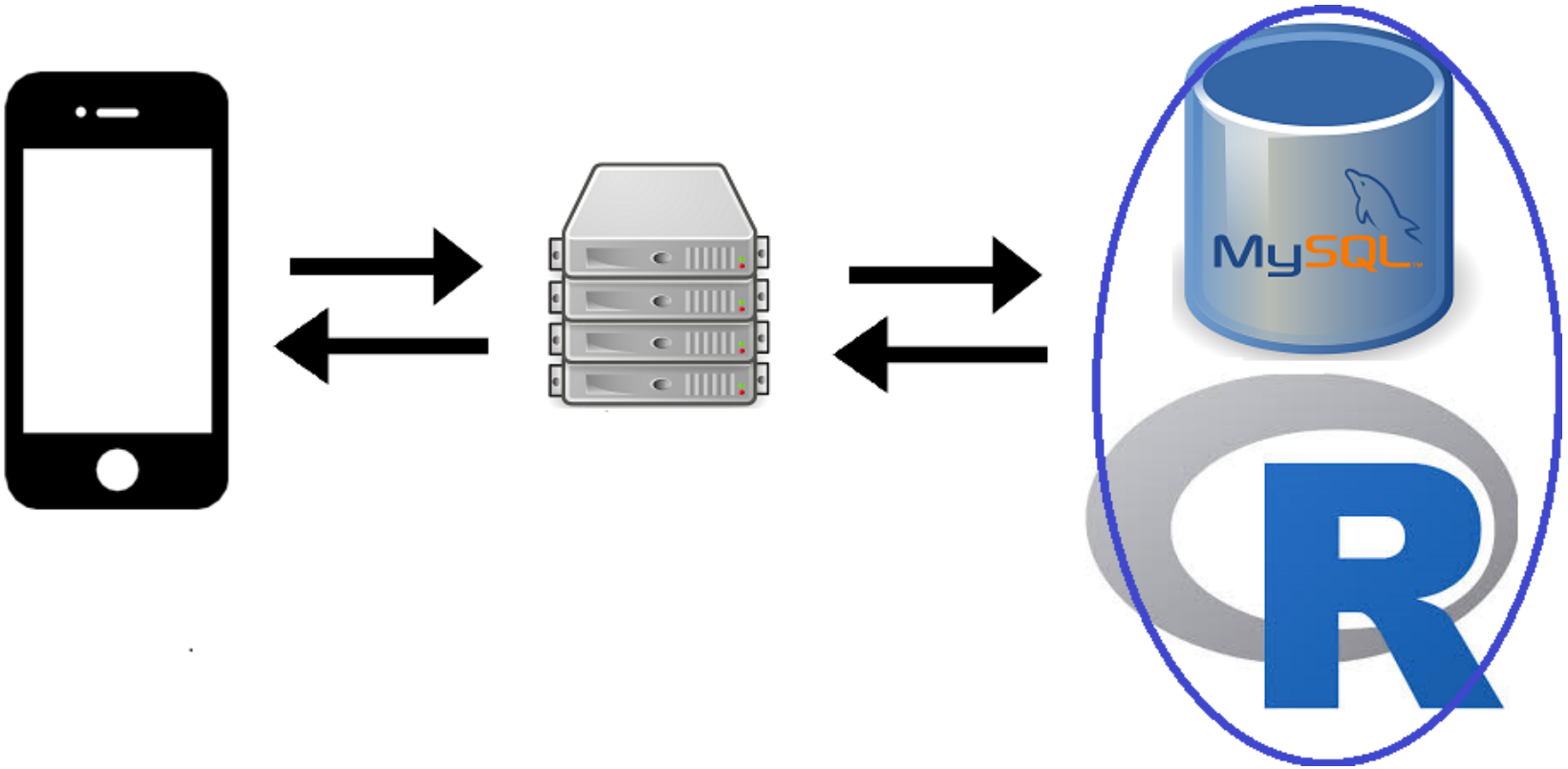


Analytics pipeline architecture: level 3

- Check out: http://anabus-surveys.com/adagppub
- R is being called by the server, after the client has requested an analysis.

Analytics pipeline architecture: level 4

R migrates to the database (in database implementation) for big data problems and immediate access to the database



Analytics pipeline architecture: level 4

Check out the

- Oracle: http://www.oracle.com/technetwork/database/database-technologies/r/r-enterprise/overview/index.html
- Microsoft: https://azure.microsoft.com/en-us/services/hdinsight/r-server/

implementations of in-database R.

Module summary

Topics covered today include:

- Data frames
- Reading data from various sources
- Database connectivity and architecture

Next time

- Merging data from different sources
- All about the regression modelling functions `lm` and `glm`

Today's function list

Do you know what each of these functions does?

```
as.character
cbind
close
data.frame
head
library
load
odbcConnect
pie
rbind
read.csv
read.table
save
sort
sqlQuery
strsplit
table
toupper
unlist
```

The MySql ODBC driver dialog

Inputs for the ODBC driver under Windows (these will connect to my database).