## Table of Contents

Term Project Proposal ........ 4
Brief Overview of Data and Need for Data Manipulation ........ 5
R ........ 6
  Basics of R ........ 6
    Variables ........ 6
    Attributes of R object ........ 6
    Vector ........ 6
    Factor ........ 7
    Data frame ........ 7
    Matrix ........ 7
    List ........ 8
    Flow control statement ........ 8
  Code in R ........ 9
  References ........ 10
Matlab ........ 11
  Basics of Matlab ........ 11
    Scalar ........ 11
    vector ........ 11
    Accessing array elements ........ 12
    Flow Control statements ........ 13
    Function ........ 15
  Code in Matlab ........ 16
  References ........ 17
MongoDB ........ 18
  Basics of MongoDB ........ 18
  Code in MongoDB ........ 18
  References ........ 23
SQL ........ 24
  Basics of SQL ........ 24

Code in SQL ........ 24
References ........ 28
Go language ........ 29
Basics of Go language ........ 29
Variables ........ 29
Constants ........ 31
Functions ........ 32
More types (of variables) ........ 33
Flow control statements ........ 36
Code in Go ........ 39
References ........ 40
Python ........ 41
Basics of Python ........ 41
Variable ........ 41
Change a case ........ 41
Slicing in Python ........ 41
Lists ........ 42
Function ........ 42
Flow Control statements ........ 43
Dictionaries ........ 44
Code in Python ........ 44
References ........ 45
Kafka ........ 46
Basics of Kafka ........ 46
Coding examples in Kafka(scala) ........ 46
References ........ 47
Comparisons of different languages ........ 48
R ........ 48
Advantages ........ 48
Disadvantages ........ 48
References(R/comparison) ........ 48
Matlab ........ 48

Advantages ........ 48
Disadvantages ........ 49
References(matlab/comparison) ........ 49
MongoDB ........ 49
Advantages ........ 49
Disadvantages ........ 49
Reference(comparison/mongoDB) ........ 50
SQL ........ 50
Advantages ........ 50
Disadvantages ........ 50
References(SQL/comparison) ........ 50
Go ........ 50
Advantages ........ 50
Disadvantages ........ 50
References(Go/comparison) ........ 51
Python ........ 51
Advantages ........ 51
Disadvantages ........ 51
References(Python/comparison) ........ 51
Kafka ........ 52
Advantages ........ 52
Disadvantage ........ 52
Reference(comparison/Kafka) ........ 52
Comments ........ 52
List of Attachments ........ 53

## Term Project Proposal

I plan to work on different languages and tools that are used in data manipulation. In particular, I will deal with R, Matlab, MongoDB, SQL, Go, Python and Kafka. (I chose to work on MongoDB instead of Hadoop after consulting with instructor).

I will start with brief overview of "data" and "need of data manipulation". I plan on exploring the basic elements of these languages and tools with few coding examples. Next I will delve into advantages and limitations of each language/tool. Lastly I will look into doing storage with few of these.

My resources would be books and online material.

## Brief Overview of Data and Need for Data Manipulation

### Overview of Big Data

Big data is a voluminous and ever-changing data, which when organized and analyzed using sophisticated softwares, provides valuable information that can be translated into improved decision making and performance. The value of big data is in the analytics and the ability to use that intelligence to gain a desired outcome. Analysis of big data marks a transformation in how society processes information.

### Need for Data Manipulation

Data manipulation is an important bridge between all the available raw data and the actual final data analysis with different tools. More often than not, the some or whole of the available raw data set is not available in the same format so as to readily allow data analysis, or can be available in a configuration different from the desired configuration, or can be available in a fragmented fashion. Data manipulation allows us to address all these issues and therefore lends significant flexibility in data collection and usage of analysis.

### References

www.hbr.org
www.foreignaffairs.com
www.whatis.techtarget.com
www.wikipedia.org
www.robertgascoyne.com/Data_Manipulation.shtml

## R

R is comprehensive statistical and graphical programming language. It was originally created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand and currently is developed by the R Development Core Team. It is a GNU project.

## Basics of R

### Variables

These are identifiers and are bound by naming rules. Objects are assigned to the variables using the ‹- operator.

### Attributes of R object

There are 2 attributes R objects have which are 'mode' and 'class'. A mode can be numeric, character or logical.
The mode indicates how a particular object has been stored in R. The mode function returns the mode of an object. For example:

```
x ‹- 232
mode(x)
>[1] "numeric"

y ‹- TRUE
mode(y)
>[1] "logical"

z ‹- ("r")
mode(z)
>[1] "character"
```

It is possible to convert mode in mixed mode objects. This is useful since mode of objects must match when they are combined in an operation.
The class of an object determines what can be done with the object. The class is assessed through class() function.

### Vector

It is a collection of values either numeric, logical or character. All the values in a vector must be of the same type. If a vector contains a single value, it is called a scalar. To create a vector, following is the command:

```
v <- c(27,1,88,65,2,10)
>v
[1] 27 1 88 65 2 10
```

## Factor

The factor type is used to encode categorical data values. These are stored as numbers internally. An example of a factor is as follows:

```
bloodGroups <- factor(c("A", "B", "AB", "O"))
>bloodGroups
[1] A B AB O
```

## Data frame

It is a tabular arrangement of rows and columns of vectors and/or factors like a spreadsheet. The vectors that make up the data frame should be of the exact same length. To create a data frame, a function called data.frame() is used which takes different vectors as its input. To create a data frame 'mixed', following is a command:

```
 numbers <- c(1,2,3,4)
letters <- c("a","b","c","d")
decimals <- c(1.1, 2.2, 3.3, 4.4)

mixed <- data.frame(numbers, letters, decimals)

>mixed
        numbers    letters    decimals
1          1          a          1.1
2          2          b          2.2
3          3          c          3.3
4          4          d          4.4
```

## Matrix

It is a 2 dimensional arrangement of data. All its elements must be of same type. To create a matrix, following command can be used:

```
m <- matrix(c(1, 0, 0,1), nrow = 2, ncol = 2, byrow = TRUE)
> m
    [,1] [,2]
[1,]   1    0
[2,]   0    1
```

Array

Array is a multi-dimensional data structure. An example of array is as follows:

```
> b <- array(1:12, c(2, 3, 2))
> b
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

## List

It can contain mixed type of objects like matrix, array, vectors and data frames. Example of how to create a list is as follows:

```
l ‹- list(b, m, mixed, bloodGroups, v)
```

The above list contains array, matrix, data frame, factor and vector respectively.

## Flow control statement

The if() statement

It is used in decision making. It guides what to do if some action is performed or some condition is set. An example is as follows:

```
 x ‹- 5
if(x > 0){
  print("Positive number")
}
```

Answer: Positive number

The ifelse() statement

It specifies what to do if the 'if statement' is not true

```
y ‹- -5
if(y > 0) print("Non-negative number") else print("Negative number")
```

Answer: Negative number

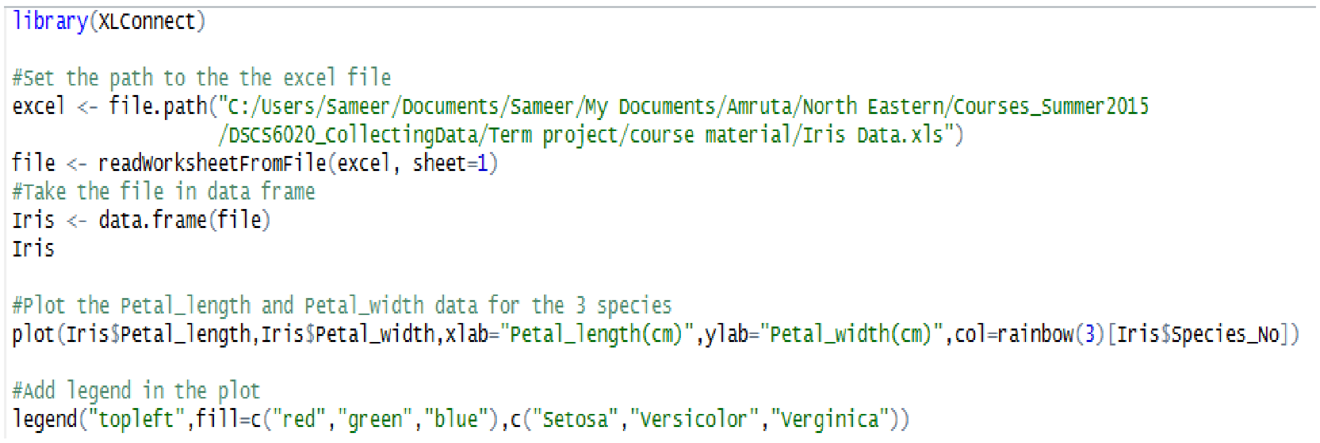
For loop
It is useful to iterate for a fixed number of times. An example is as follows:

```
>  samples <- c(rep(1:10))
> for(sample in samples){print (sample)}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

## Code in R

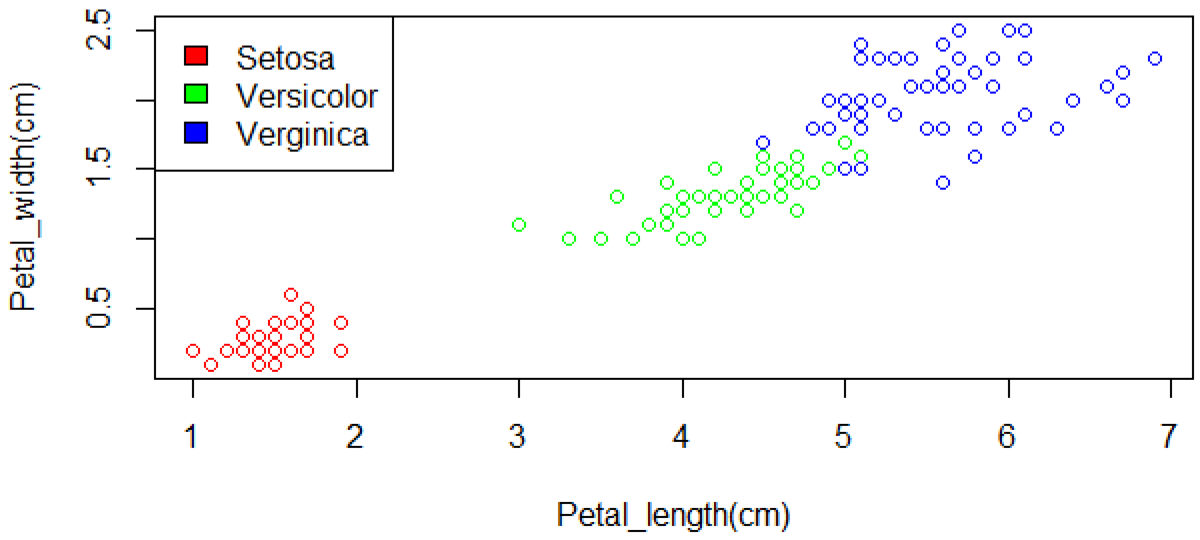
To apply the R in terms of  collection, storage and retrieval of data, I used a file named 'Iris Data.xls'. First I took, 'Iris Data.xls' file into R in the form of data frame. Then to retrieve a data from it, I used a plot (I have drawn the same plot using Matlab). All these operations are done in a file called **FisherIris.R**

Following is the program code in R:
library(XLConnect)



```
library(XLConnect)

#Set the path to the the excel file
excel <- file.path("C:/Users/Sameer/Documents/Sameer/My Documents/Amruta/North Eastern/Courses_Summer2015
                   /DSCS6020_CollectingData/Term project/course material/Iris Data.xls")
file <- readWorksheetFromFile(excel, sheet=1)
#Take the file in data frame
Iris <- data.frame(file)
Iris

#Plot the Petal_length and Petal_width data for the 3 species
plot(Iris$Petal_length,Iris$Petal_width,xlab="Petal_length(cm)",ylab="Petal_width(cm)",col=rainbow(3)[Iris$Species_No])

#Add legend in the plot
legend("topleft",fill=c("red","green","blue"),c("Setosa","Versicolor","Verginica"))
```

The output of above R code is as follows:



## References

https://en.wikipedia.org/wiki/R_(programming_language)
DSCS6020 course material

# Matlab

## Basics of Matlab

### Scalar

A scalar variable is declared as follows:

```
B=2
```

### vector

It contains only one row or only one column. An example is as follows:

```
v =0:1:5


v =


     0     1     2     3     4     5
```

```
d = [10; 20; 30]


d =


    10

    20

    30
```

In matlab, columns are separated by ';' in the command.

A matrix is a 2-dimentional, rectangular array. To create a matrix [] operator can be used. A matrix can be created as follows:

```
B = [12, 62, 93, -8, 22; 16, 2, 87, 43, 91; -4, 17, -72, 95, 6]
```

```
B =


    12    62    93    -8    22

    16     2    87    43    91

    -4    17   -72    95     6
```

Another way of creating a matrix is as follows:

```
A = [0 1 2 3 4 5 5; 13 12 16 10 10 20 19]


A =


     0     1     2     3     4     5     5

    13    12    16    10    10    20    19
```

An array with more than 2 dimensions is called a multidimensional array.

## Accessing array elements

In the above example, to access an element at row 4 column 2 of matrix B can be done as follows:

```
B(3,2)


ans =


    17
```

To access some part of array, array slicing can be done as follows:

```
A(1:6)

ans =
     0    13     1    12     2    16
```

```
A(1:2,2:4)

ans =

     1     2     3
    12    16    10
```

To take entire second row and all the columns, following is the command:

```
A(2,:)

ans =

    13    12    16    10    10    20    19
```

To take entire second column and all the rows, following is the command:

```
A(:,2)

ans =

     1
    12
```

## Flow Control statements

if statement
Syntax and example of if statement is as follows:

A random number is generated:

```
a = randi(100, 1)
% If it is even, divide by 2
if rem(a, 2) == 0
    disp('a is even')
    b = a/2
end
```

In matlab, comments are preceded by '%' symbol if else statement

Following is the example and syntax for if else statement. First a user input is taken.

```
yourNumber = input('Enter a number: ');


if yourNumber < 0
   disp('Negative')
elseif yourNumber > 0
   disp('Positive')
else
   disp('Zero')
end


yourNumber = 8
answer: Positive
```

For statement

It loops a specific number of times. An example is as follows:

```
x = ones(1,10)
x =


     1     1     1     1     1     1     1     1     1     1
for n = 2:3
   x(n) = 2 * x(n - 1)
end


x =


     1     2     4     8    16    32     1     1     1     1
```

```
x =

     1     2     4     8    16    32     1     1     1     1
```

while statement loops as long as a condition remains true.

```
p=10;

while (p<14)

fprintf ('value of p: %d\n',p);

p =p+1;

end
```

Answer:

```
value of p: 10

value of p: 11

value of p: 12

value of p: 13
```

## Function

The example is as follows:

```
function y = average(x)
if ~isvector(x)
error('Input must be a vector')
end
y = sum(x)/length(x);
end
```

A call to the function is as follows:

```
z=1:99;
average(z)
```

answer: 50

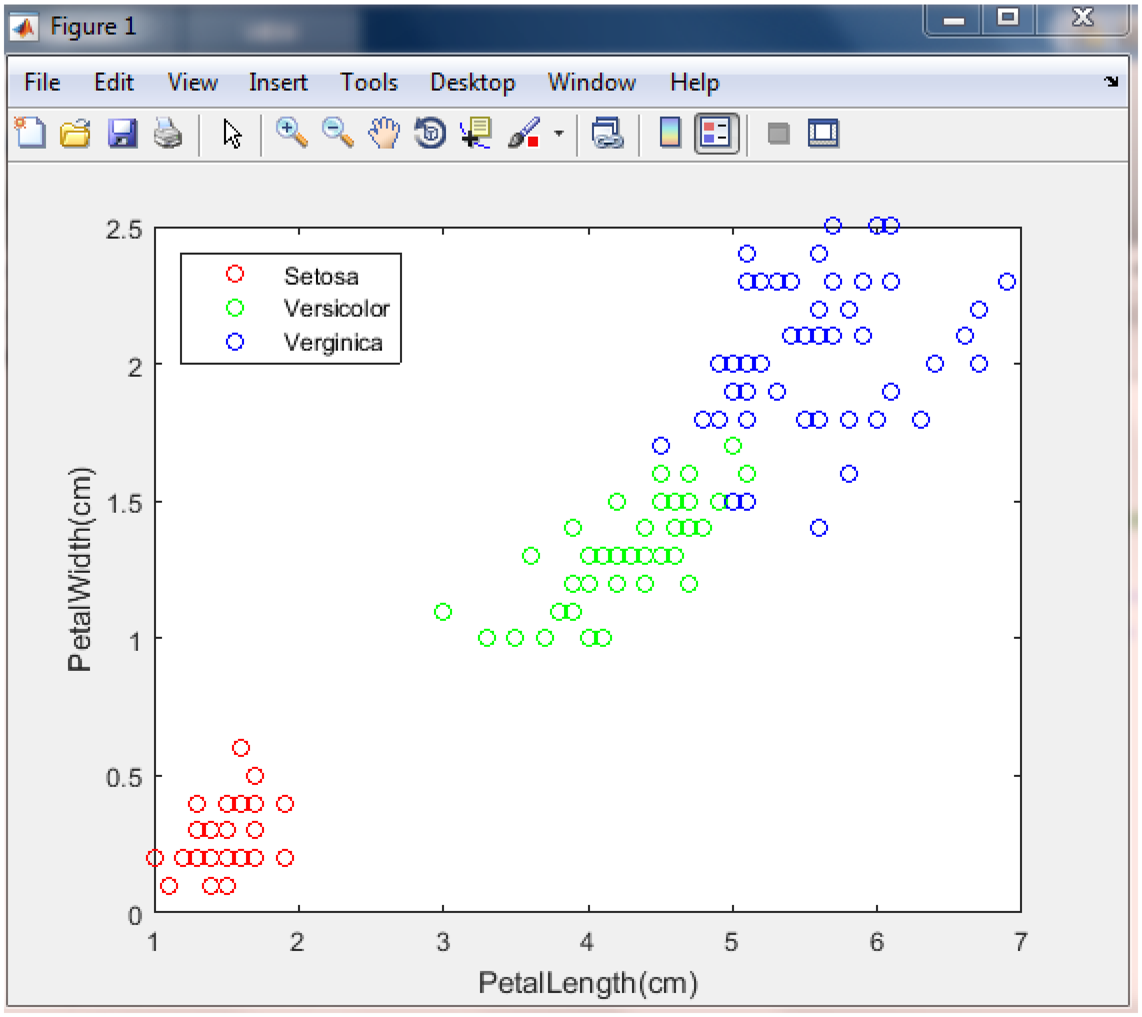
## Code in Matlab

To apply the matlab in terms of  collection, storage and retrieval of data, I used a file named 'Iris Data.xls'. First I took, 'Iris Data.xls' file into matlab. To do this I used a command readtable('Iris Data.xls'). I stored  it in a table form and saved it in a variable called 'flowerInfo'. Then to retrieve a data from it, I used a plot (I have drawn the same plot using R). All these operations are done in a file called **FisherIris.m**
Please note that I have stored this file in a 'table' data type since it is useful to collect mixed-type data and metadata properties like variable name, row names, descriptions and variable units in a single container.

Following is the program code in matlab:

```
%The xls file Iris Data.xls is taken in a table format and saved in a
%variable called flowerInfo.
flowerInfo = readtable('Iris Data.xls');

%I have created 3 variables which use string compare function for
%conditional data retrieval
ID1 = strcmp(cellstr(flowerInfo.Species_name),' Setosa');
ID2 = strcmp(cellstr(flowerInfo.Species_name),' Versicolor');
ID3 = strcmp(cellstr(flowerInfo.Species_name),' Verginica');

%Following is the command to retrive the data from a table in the form of a
%plot
plot(flowerInfo.Petal_length(ID1),flowerInfo.Petal_width(ID1),'or',...
    flowerInfo.Petal_length(ID2),flowerInfo.Petal_width(ID2),'og',...
    flowerInfo.Petal_length(ID3),flowerInfo.Petal_width(ID3),'ob');

%The x and y axis labels are defined as follows:
xlabel('PetalLength(cm)')
ylabel('PetalWidth(cm)')

%The legend is added with the following command:
legend('Setosa','Versicolor','Verginica')
%The location of the legend is specified as follows:
legend('Location','Northwest')
```

The output of above matlab code is as follows:



## References

http://www.mathworks.com/help

## MongoDB

MongoDB is a NoSQL database. It is document oriented type of database. It was first developed by the software company MongoDBInc. It stores data in the form of 'documents' which are JSON-like field and value pairs. Formally the documents are BSON documents.

### Basics of MongoDB

CRUD operations in MongoDB
It stores all documents in collections. A collection is a group of related documents that have a set of shared common indexes. A collection is equivalent to RDBMS table. It uses map reduce which is a data processing paradigm for condensing large volumes of data into useful aggregated results.
Query- Query or read operation targets a single, specific collection of documents. Using query fields from documents can be fetched. The queries can be manipulated to impose limits, skips and sort orders. Some query examples are as follows:

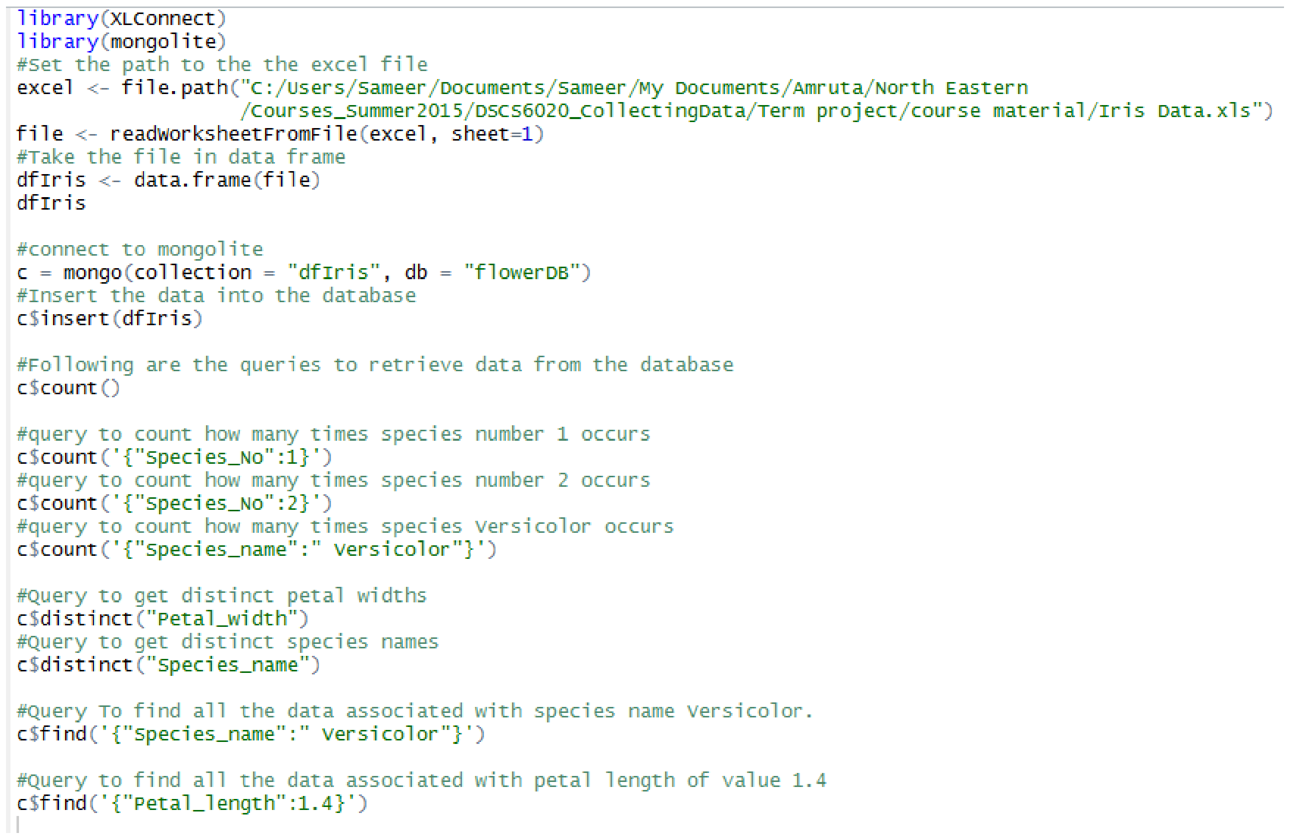
This query selects names and addresses of people greater than age 18. It returns at most 5 matching documents.
db.users.find( { age: { $gt: 18 } }, { name: 1, address: 1 } ).limit(5)

The following query excludes one field from a result set.
db.records.find( { "user_id": { $lt: 42 } }, { "history": 0 } )

This query returns 2 fields and the _id field
db.records.find( { "user_id": { $lt: 42 } }, { "name": 1, "email": 1 } )

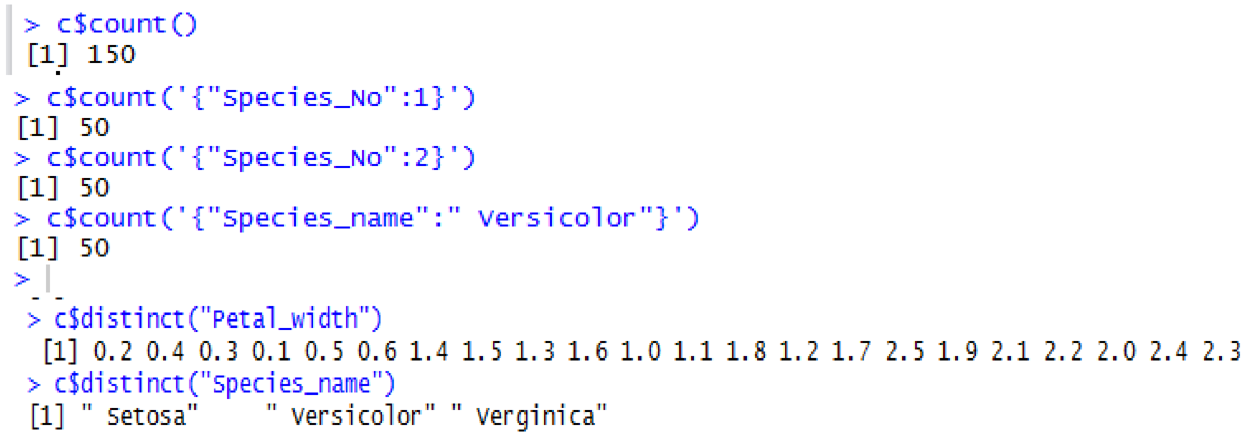
### Code in MongoDB

I have used mongolite package in R. To apply mongolite in terms of collection, storage and retrieval, here also I used excel file **'Iris Data.xls'**. After storing it in the database, I used some queries to retrieve the data. The program named '**submission_TermProject_mongoDB_Iris.R**' is as follows:



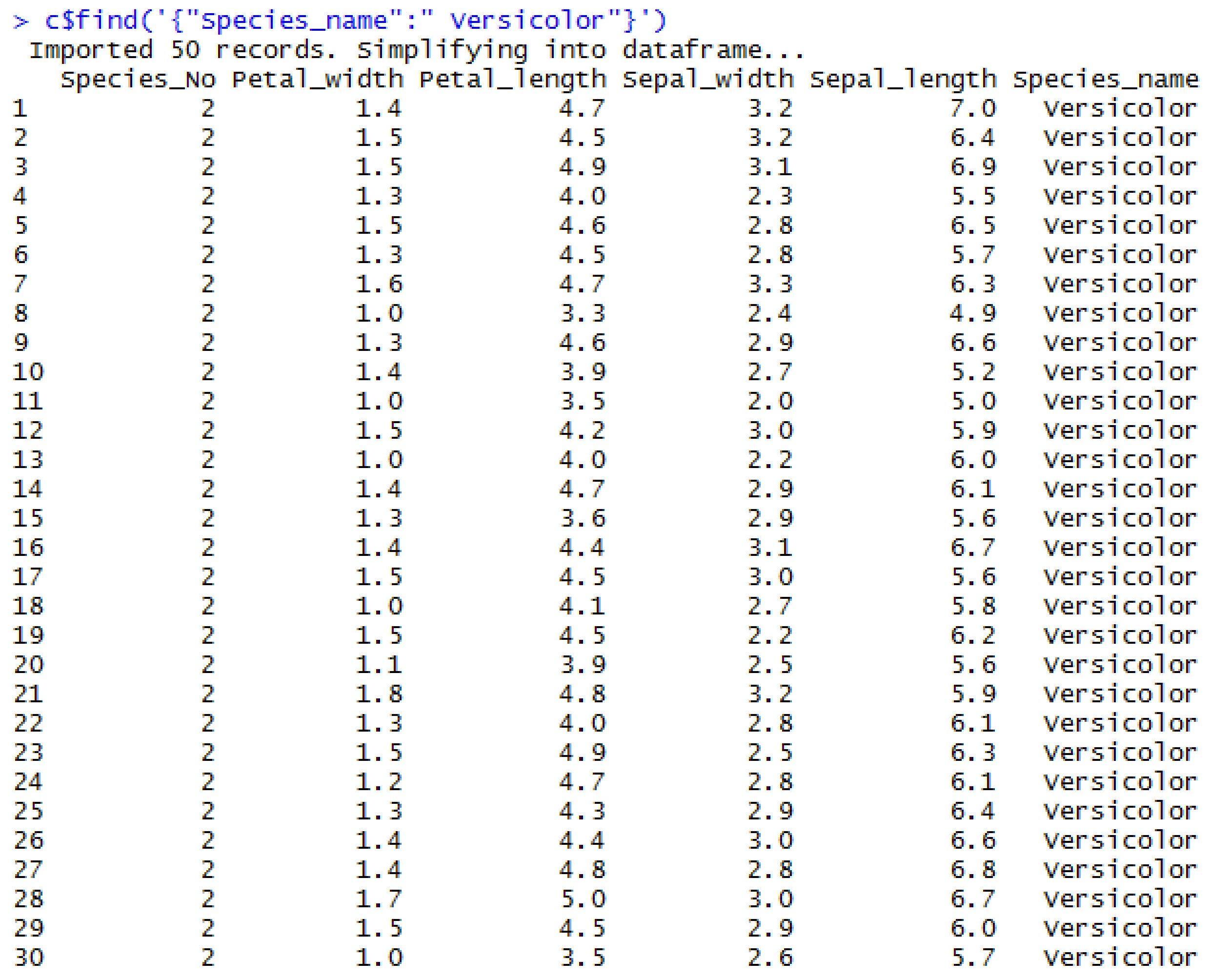
```
library(XLConnect)
library(mongolite)
#Set the path to the the excel file
excel <- file.path("C:/Users/Sameer/Documents/Sameer/My Documents/Amruta/North Eastern
                   /Courses_Summer2015/DSCS6020_CollectingData/Term project/course material/Iris Data.xls")
file <- readWorksheetFromFile(excel, sheet=1)
#Take the file in data frame
dfIris <- data.frame(file)
dfIris

#connect to mongolite
c = mongo(collection = "dfIris", db = "flowerDB")
#Insert the data into the database
c$insert(dfIris)

#Following are the queries to retrieve data from the database
c$count()

#query to count how many times species number 1 occurs
c$count('{"Species_No":1}')
#query to count how many times species number 2 occurs
c$count('{"Species_No":2}')
#query to count how many times species Versicolor occurs
c$count('{"Species_name":" Versicolor"}')

#Query to get distinct petal widths
c$distinct("Petal_width")
#Query to get distinct species names
c$distinct("Species_name")

#Query To find all the data associated with species name Versicolor.
c$find('{"Species_name":" Versicolor"}')

#Query to find all the data associated with petal length of value 1.4
c$find('{"Petal_length":1.4}')
```

Here is the output of the queries:

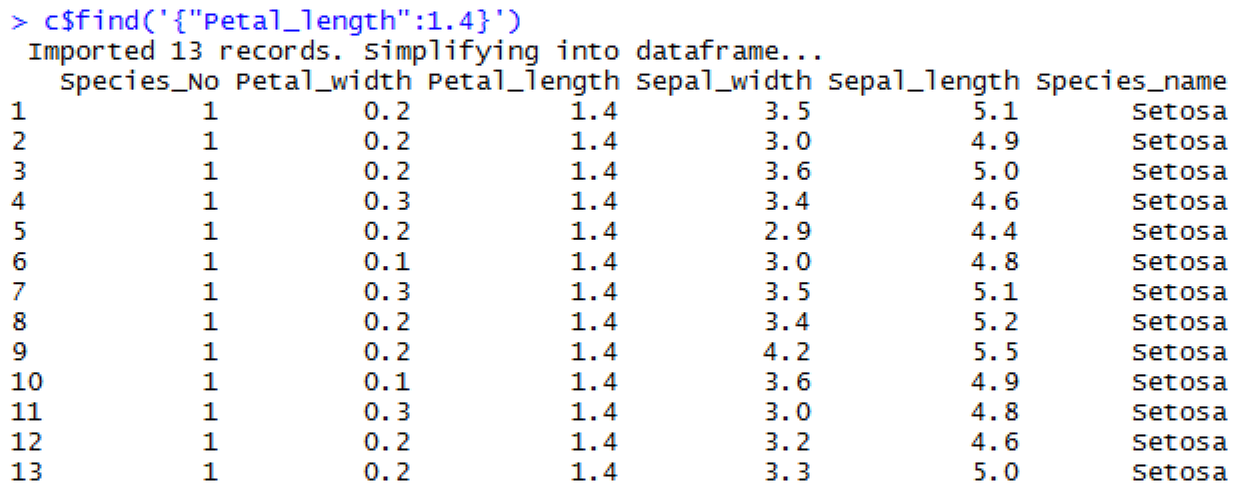


```
> c$count()
[1] 150
> c$count('{"Species_No":1}')
[1] 50
> c$count('{"Species_No":2}')
[1] 50
> c$count('{"Species_name":" Versicolor"}')
[1] 50
>
> c$distinct("Petal_width")
 [1] 0.2 0.4 0.3 0.1 0.5 0.6 1.4 1.5 1.3 1.6 1.0 1.1 1.8 1.2 1.7 2.5 1.9 2.1 2.2 2.0 2.4 2.3
> c$distinct("Species_name")
[1] " Setosa"     " Versicolor" " Verginica"
```

Following are few lines of output of the following query. I am showing only some lines since the output has many rows.



```
> c$find('{"Species_name":" Versicolor"}')
 Imported 50 records. Simplifying into dataframe...
   Species_No Petal_width Petal_length Sepal_width Sepal_length Species_name
1           2         1.4          4.7         3.2          7.0   Versicolor
2           2         1.5          4.5         3.2          6.4   Versicolor
3           2         1.5          4.9         3.1          6.9   Versicolor
4           2         1.3          4.0         2.3          5.5   Versicolor
5           2         1.5          4.6         2.8          6.5   Versicolor
6           2         1.3          4.5         2.8          5.7   Versicolor
7           2         1.6          4.7         3.3          6.3   Versicolor
8           2         1.0          3.3         2.4          4.9   Versicolor
9           2         1.3          4.6         2.9          6.6   Versicolor
10          2         1.4          3.9         2.7          5.2   Versicolor
11          2         1.0          3.5         2.0          5.0   Versicolor
12          2         1.5          4.2         3.0          5.9   Versicolor
13          2         1.0          4.0         2.2          6.0   Versicolor
14          2         1.4          4.7         2.9          6.1   Versicolor
15          2         1.3          3.6         2.9          5.6   Versicolor
16          2         1.4          4.4         3.1          6.7   Versicolor
17          2         1.5          4.5         3.0          5.6   Versicolor
18          2         1.0          4.1         2.7          5.8   Versicolor
19          2         1.5          4.5         2.2          6.2   Versicolor
20          2         1.1          3.9         2.5          5.6   Versicolor
21          2         1.8          4.8         3.2          5.9   Versicolor
22          2         1.3          4.0         2.8          6.1   Versicolor
23          2         1.5          4.9         2.5          6.3   Versicolor
24          2         1.2          4.7         2.8          6.1   Versicolor
25          2         1.3          4.3         2.9          6.4   Versicolor
26          2         1.4          4.4         3.0          6.6   Versicolor
27          2         1.4          4.8         2.8          6.8   Versicolor
28          2         1.7          5.0         3.0          6.7   Versicolor
29          2         1.5          4.5         2.9          6.0   Versicolor
30          2         1.0          3.5         2.6          5.7   Versicolor
```

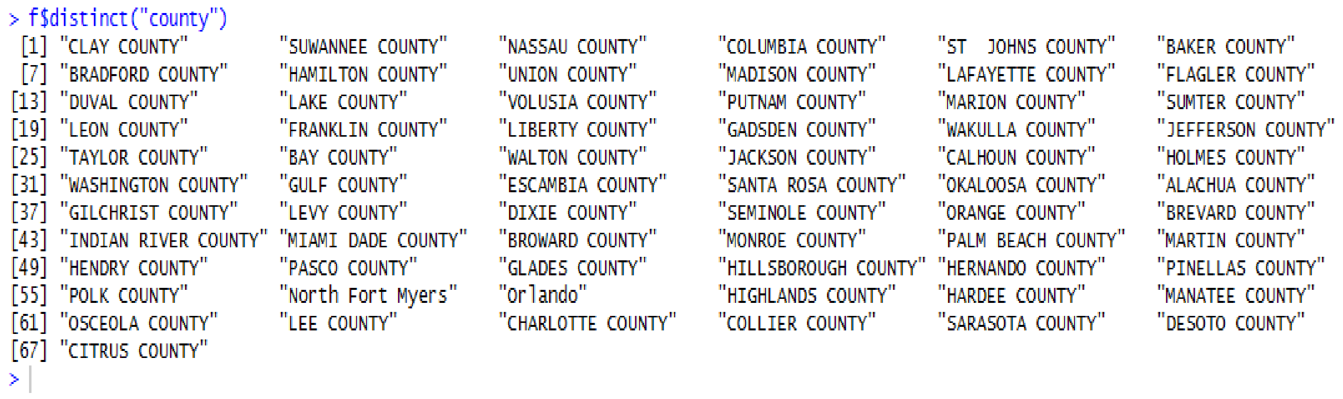


```
> c$find('{"Petal_length":1.4}')
 Imported 13 records. Simplifying into dataframe...
   Species_No Petal_width Petal_length Sepal_width Sepal_length Species_name
1           1         0.2          1.4         3.5          5.1       Setosa
2           1         0.2          1.4         3.0          4.9       Setosa
3           1         0.2          1.4         3.6          5.0       Setosa
4           1         0.3          1.4         3.4          4.6       Setosa
5           1         0.2          1.4         2.9          4.4       Setosa
6           1         0.1          1.4         3.0          4.8       Setosa
7           1         0.3          1.4         3.5          5.1       Setosa
8           1         0.2          1.4         3.4          5.2       Setosa
9           1         0.2          1.4         4.2          5.5       Setosa
10          1         0.1          1.4         3.6          4.9       Setosa
11          1         0.3          1.4         3.0          4.8       Setosa
12          1         0.2          1.4         3.2          4.6       Setosa
13          1         0.2          1.4         3.3          5.0       Setosa
```

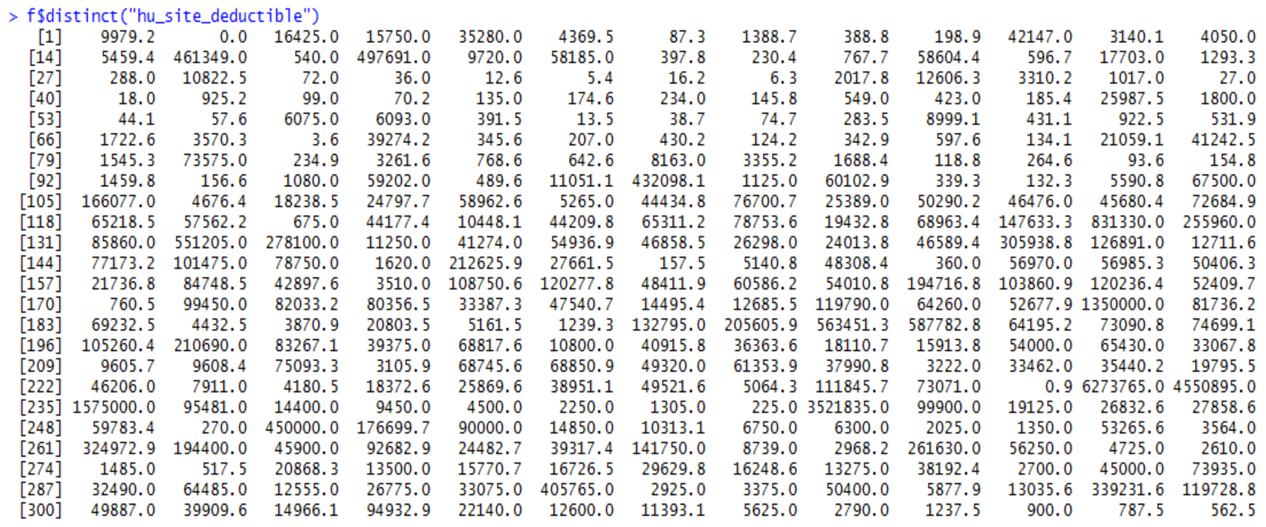
In addition to the 'Iris Data.xls' file I have taken **'FL_insurance_sample.csv'** file into mongolite and have retrieved data from it through queries. I used this file to collect, store and retrieve some big data file in mongolite. The program **'submission_TermProject_mongoDB_FLinsurance.R'** is as follows:

```
#A path was set to the destination file using setwd()
setwd("C:/Users/Sameer/Documents/Sameer/My Documents/Amruta/North Eastern
      /Courses_Summer2015/DSCS6020_CollectingData/Term project")

#The csv file is read into a data frame and stored as "FLinsurance"
FLinsurance <- read.csv("FL_insurance_sample.csv",fill=TRUE,header=TRUE,sep=",")

#Load the library mongolite
library(mongolite)

#Connect to mongolite
f = mongo(collection = "FLinsurance", db = "FLinsuranceDB")
#Insert the data into the database
f$insert(FLinsurance)

#Following are the queries to retrieve data from the database


#Show distinct "county" from the data
f$distinct("county")
#Show distinct "hu_site_deductible" from the data
f$distinct("hu_site_deductible")

#Count all the "CLAY COUNTY" entries from the column county
f$count('{"county":"CLAY COUNTY"}')
#Count all the "Wood" entries from the column construction
f$count('{"construction":"Wood"}')

#Query to find all the data associated with "policyID" of value 119736
f$find('{"policyID":119736}')
```

Following are the output of queries:

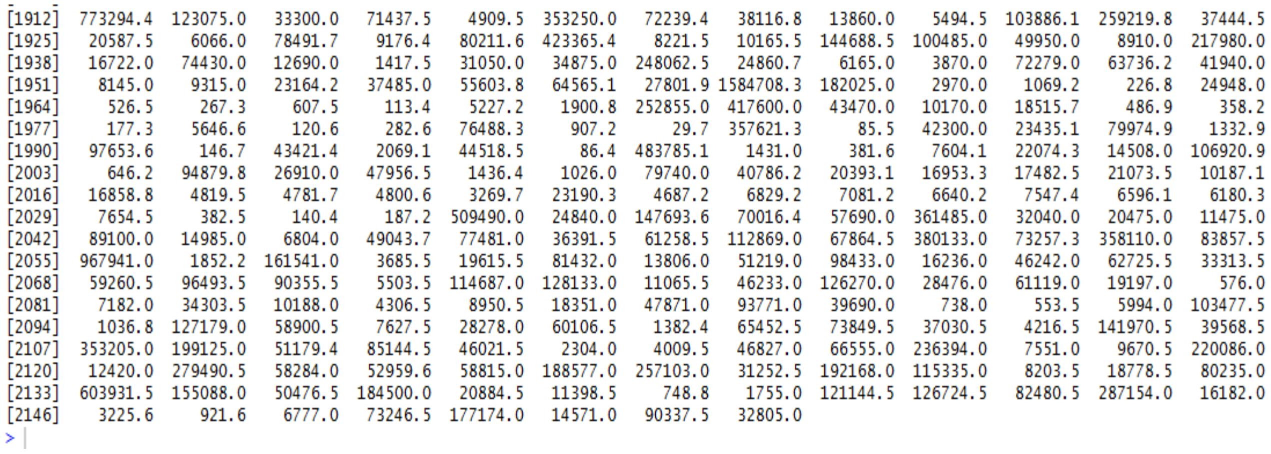


```
> f$distinct("county")
 [1] "CLAY COUNTY"         "SUWANNEE COUNTY"    "NASSAU COUNTY"      "COLUMBIA COUNTY"     "ST  JOHNS COUNTY"   "BAKER COUNTY"
 [7] "BRADFORD COUNTY"     "HAMILTON COUNTY"    "UNION COUNTY"       "MADISON COUNTY"      "LAFAYETTE COUNTY"   "FLAGLER COUNTY"
[13] "DUVAL COUNTY"        "LAKE COUNTY"        "VOLUSIA COUNTY"     "PUTNAM COUNTY"       "MARION COUNTY"      "SUMTER COUNTY"
[19] "LEON COUNTY"         "FRANKLIN COUNTY"    "LIBERTY COUNTY"     "GADSDEN COUNTY"      "WAKULLA COUNTY"     "JEFFERSON COUNTY"
[25] "TAYLOR COUNTY"       "BAY COUNTY"         "WALTON COUNTY"      "JACKSON COUNTY"      "CALHOUN COUNTY"     "HOLMES COUNTY"
[31] "WASHINGTON COUNTY"   "GULF COUNTY"        "ESCAMBIA COUNTY"    "SANTA ROSA COUNTY"   "OKALOOSA COUNTY"    "ALACHUA COUNTY"
[37] "GILCHRIST COUNTY"    "LEVY COUNTY"        "DIXIE COUNTY"       "SEMINOLE COUNTY"     "ORANGE COUNTY"      "BREVARD COUNTY"
[43] "INDIAN RIVER COUNTY" "MIAMI DADE COUNTY"  "BROWARD COUNTY"     "MONROE COUNTY"       "PALM BEACH COUNTY"  "MARTIN COUNTY"
[49] "HENDRY COUNTY"       "PASCO COUNTY"       "GLADES COUNTY"      "HILLSBOROUGH COUNTY" "HERNANDO COUNTY"    "PINELLAS COUNTY"
[55] "POLK COUNTY"         "North Fort Myers"   "Orlando"            "HIGHLANDS COUNTY"    "HARDEE COUNTY"      "MANATEE COUNTY"
[61] "OSCEOLA COUNTY"      "LEE COUNTY"         "CHARLOTTE COUNTY"   "COLLIER COUNTY"      "SARASOTA COUNTY"    "DESOTO COUNTY"
[67] "CITRUS COUNTY"
>
```

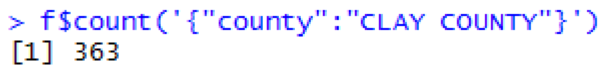


```
> f$distinct("hu_site_deductible")
   [1]      9979.2       0.0   16425.0   15750.0   35280.0    4369.5      87.3    1388.7     388.8     198.9   42147.0    3140.1    4050.0
  [14]      5459.4  461349.0     540.0  497691.0    9720.0   58185.0     397.8     230.4     767.7   58604.4     596.7   17703.0    1293.3
  [27]       288.0   10822.5      72.0      36.0      12.6       5.4      16.2       6.3    2017.8   12606.3    3310.2    1017.0      27.0
  [40]        18.0     925.2      99.0      70.2     135.0     174.6     234.0     145.8     549.0     423.0     185.4   25987.5    1800.0
  [53]        44.1      57.6    6075.0    6093.0     391.5      13.5      38.7      74.7     283.5    8999.1     431.1     922.5     531.9
  [66]      1722.6    3570.3       3.6   39274.2     345.6     207.0     430.2     124.2     342.9     597.6     134.1   21059.1   41242.5
  [79]      1545.3   73575.0     234.9    3261.6     768.6     642.6    8163.0    3355.2    1688.4     118.8     264.6      93.6     154.8
  [92]      1459.8     156.6    1080.0   59202.0     489.6   11051.1  432098.1    1125.0   60102.9     339.3     132.3    5590.8   67500.0
 [105]    166077.0    4676.4   18238.5   24797.7   58962.6    5265.0   44434.8   76700.7   25389.0   50290.2   46476.0   45680.4   72684.9
 [118]     65218.5   57562.2     675.0   44177.4   10448.1   44209.8   65311.2   78753.6   19432.8   68963.4  147633.3  831330.0  255960.0
 [131]     85860.0  551205.0  278100.0   11250.0   41274.0   54936.9   46858.5   26298.0   24013.8   46589.4  305938.8  126891.0   12711.6
 [144]     77173.2  101475.0   78750.0    1620.0  212625.9   27661.5     157.5    5140.8   48308.4     360.0   56970.0   56985.3   50406.3
 [157]     21736.8   84748.5   42897.6    3510.0  108750.6  120277.8   48411.9   60586.2   54010.8  194716.8  103860.9  120236.4   52409.7
 [170]       760.5   99450.0   82033.2   80356.5   33387.3   47540.7   14495.4   12685.5  119790.0   64260.0   52677.9 1350000.0   81736.2
 [183]     69232.5    4432.5    3870.9   20803.5    5161.5    1239.3  132795.0  205605.9  563451.3  587782.8   64195.2   73090.8   74699.1
 [196]    105260.4  210690.0   83267.1   39375.0   68817.6   10800.0   40915.8   36363.6   18110.7   15913.8   54000.0   65430.0   33067.8
 [209]      9605.7    9608.4   75093.3    3105.9   68745.6   68850.9   49320.0   61353.9   37990.8    3222.0   33462.0   35440.2   19795.5
 [222]     46206.0    7911.0    4180.5   18372.6   25869.6   38951.1   49521.6    5064.3  111845.7   73071.0       0.9 6273765.0 4550895.0
 [235]   1575000.0   95481.0   14400.0    9450.0    4500.0    2250.0    1305.0     225.0 3521835.0   99900.0   19125.0   26832.6   27858.6
 [248]     59783.4     270.0  450000.0  176699.7   90000.0   14850.0   10313.1    6750.0    6300.0    2025.0    1350.0   53265.6    3564.0
 [261]    324972.9  194400.0   45900.0   92682.9   24482.7   39317.4  141750.0    8739.0    2968.2  261630.0   56250.0    4725.0    2610.0
 [274]      1485.0     517.5   20868.3   13500.0   15770.7   16726.5   29629.8   16248.6   13275.0   38192.4    2700.0   45000.0   73935.0
 [287]     32490.0   64485.0   12555.0   26775.0   33075.0  405765.0    2925.0    3375.0   50400.0    5877.9   13035.6  339231.6  119728.8
 [300]     49887.0   39909.6   14966.1   94932.9   22140.0   12600.0   11393.1    5625.0    2790.0    1237.5     900.0     787.5     562.5
```

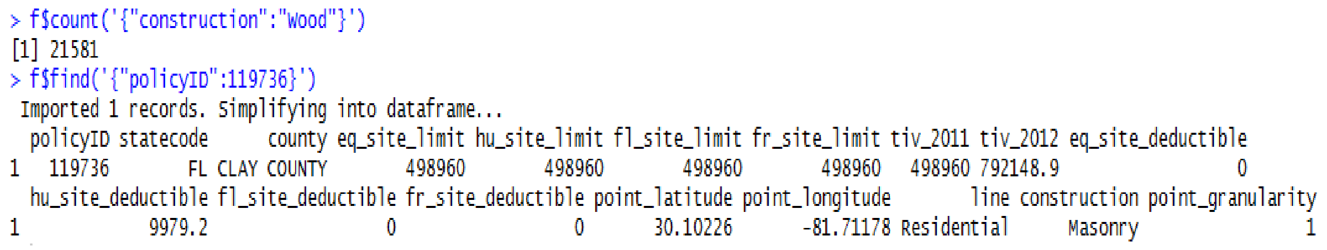
Here are last few lines of the above query. I have not included all the rows since there are many number of rows.



```
[1912]  773294.4  123075.0   33300.0   71437.5    4909.5  353250.0   72239.4   38116.8   13860.0    5494.5  103886.1  259219.8   37444.5
[1925]   20587.5    6066.0   78491.7    9176.4   80211.6  423365.4    8221.5   10165.5  144688.5  100485.0   49950.0    8910.0  217980.0
[1938]   16722.0   74430.0   12690.0    1417.5   31050.0   34875.0  248062.5   24860.7    6165.0    3870.0   72279.0   63736.2   41940.0
[1951]    8145.0    9315.0   23164.2   37485.0   55603.8   64565.1   27801.9 1584708.3  182025.0    2970.0    1069.2     226.8   24948.0
[1964]     526.5     267.3     607.5     113.4    5227.2    1900.8  252855.0  417600.0   43470.0   10170.0   18515.7     486.9     358.2
[1977]     177.3    5646.6     120.6     282.6   76488.3     907.2      29.7  357621.3      85.5   42300.0   23435.1   79974.9    1332.9
[1990]   97653.6     146.7   43421.4    2069.1   44518.5      86.4  483785.1    1431.0     381.6    7604.1   22074.3   14508.0  106920.9
[2003]     646.2   94879.8   26910.0   47956.5    1436.4    1026.0   79740.0   40786.2   20393.1   16953.3   17482.5   21073.5   10187.1
[2016]   16858.8    4819.5    4781.7    4800.6    3269.7   23190.3    4687.2    6829.2    7081.2    6640.2    7547.4    6596.1    6180.3
[2029]    7654.5     382.5     140.4     187.2  509490.0   24840.0  147693.6   70016.4   57690.0  361485.0   32040.0   20475.0   11475.0
[2042]   89100.0   14985.0    6804.0   49043.7   77481.0   36391.5   61258.5  112869.0   67864.5  380133.0   73257.3  358110.0   83857.5
[2055]  967941.0    1852.2  161541.0    3685.5   19615.5   81432.0   13806.0   51219.0   98433.0   16236.0   46242.0   62725.5   33313.5
[2068]   59260.5   96493.5   90355.5    5503.5  114687.0  128133.0   11065.5   46233.0  126270.0   28476.0   61119.0   19197.0     576.0
[2081]    7182.0   34303.5   10188.0    4306.5    8950.5   18351.0   47871.0   93771.0   39690.0     738.0     553.5    5994.0  103477.5
[2094]    1036.8  127179.0   58900.5    7627.5   28278.0   60106.5    1382.4   65452.5   73849.5   37030.5    4216.5  141970.5   39568.5
[2107]  353205.0  199125.0   51179.4   85144.5   46021.5    2304.0    4009.5   46827.0   66555.0  236394.0    7551.0    9670.5  220086.0
[2120]   12420.0  279490.5   58284.0   52959.6   58815.0  188577.0  257103.0   31252.5  192168.0  115335.0    8203.5   18778.5   80235.0
[2133]  603931.5  155088.0   50476.5  184500.0   20884.5   11398.5     748.8    1755.0  121144.5  126724.5   82480.5  287154.0   16182.0
[2146]    3225.6     921.6    6777.0   73246.5  177174.0   14571.0   90337.5   32805.0
> |
```



```
> f$count('{"county":"CLAY COUNTY"}')
[1] 363
```



```
> f$count('{"construction":"Wood"}')
[1] 21581
> f$find('{"policyID":119736}')
 Imported 1 records. Simplifying into dataframe...
  policyID statecode      county eq_site_limit hu_site_limit fl_site_limit fr_site_limit tiv_2011 tiv_2012 eq_site_deductible
1   119736        FL CLAY COUNTY        498960        498960        498960        498960   498960 792148.9                  0
  hu_site_deductible fl_site_deductible fr_site_deductible point_latitude point_longitude        line construction point_granularity
1             9979.2                  0                  0       30.10226       -81.71178 Residential      Masonry                 1
```

## References

https://en.wikipedia.org/wiki/MongoDB
http://docs.mongodb.org/manual/core/data-model-design/

## SQL

SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in 1970s. It is designed for managing data held in relational database management system. It has become standard of the American National Standard Institute (ANSI) in 1986 for International Organization for Standardization (ISO) in 1987. There are several elements of SQL, including clauses, expressions, predicates, queries, and statements.

### Basics of SQL

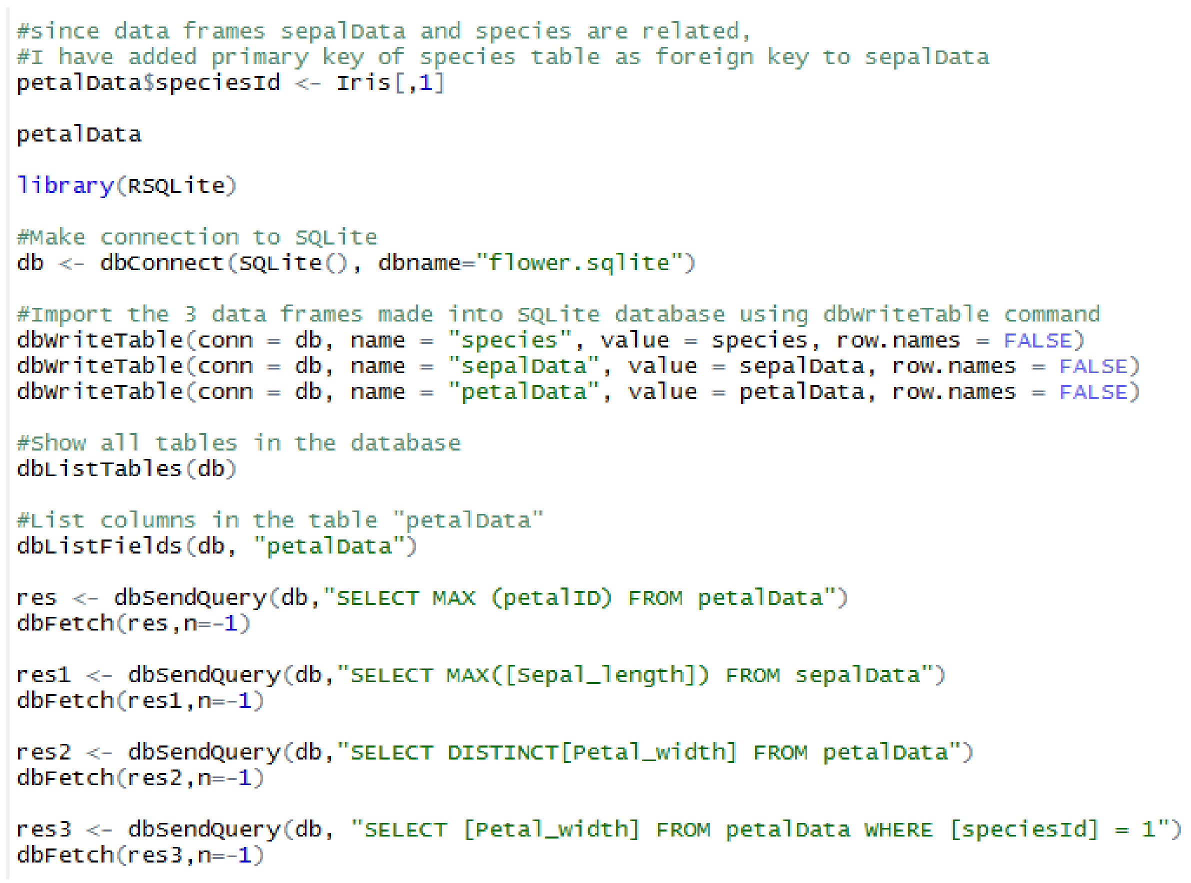
The queries are important part this language. The project operation are used to specify which columns to work with. The most common operation in this language are the ones carried out by using SELECT statements. Several other clauses are used in SELECT statement which allow the user to retrieve specific portion from a table or from several tables. Following are some of those clauses:
The FROM clause: This tells the SELECT statement from where the data should be retrieved.
The WHERE clause: This is useful for the comparison purpose, which eliminates the rows from the table which do not match the specified criteria .
The GROUP BY clause: It is used to project rows with common values into a smaller set of rows.
The HAVING clause: This is used to further filter the rows that result from the GROUP BY clause.
The ORDER BY clause: This is useful in sorting the order of columns obtained in ascending or descending order.
Below are some examples that show the use of these clauses:

```
SELECT store_name from store_information
SELECT * FROM departments
SELECT store_name FROM store_information WHERE productSale>4300
SELECT store_name, SUM(productSale) FROM store_information GROUP BY store_name
SELECT store_name, SUM(productSale) FROM store_information GROUP BY store_name
HAVING SUM(productSale) > 5600
SELECT store_name, productSale, Txn_date FROM store_information ORDER BY productSale
DESC
```

### Code in SQL

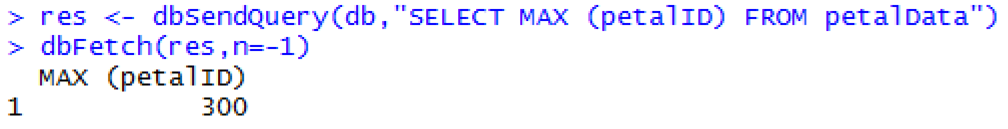
For the SQL language application, I took Fisher Iris data from a file 'Iris Data.xls' then after identifying individual entities from the data I subdivided that into 3 tables called species, sepalData and petalData by manipulating it in R. I took that data in SQLite and created a database. Further I wrote some queries to retrieve that data stored in tables. Following is my program:

```
library(XLConnect)

#Set the path to the the excel file
excel <- file.path("C:/Users/Sameer/Documents/Sameer/My Documents/Amruta/North Eastern/Courses_Summer2015
                   /DSCS6020_CollectingData/Term project/course material/Iris Data.xls")
file <- readWorksheetFromFile(excel, sheet=1)
#Take the file in data frame
Iris <- data.frame(file)
Iris

#From the data frame Iris, I need to make separate data frames to so that it will be easy to create
#relationships between different entities.
#A data frame species is created which contains species ids and species names.
species <- Iris[,c(1,6)]
species

#A data frame called sepalData is initialized as follows
sepalData <- Iris[,c(4,5)]

#A primary key is added to data frame sepalData by using seq() function
sepalData$sepalId <- seq(1,150,by=1)

#since data frames sepalData and species are related, I have added primary key of species table
#as foreign key to sepalData
sepalData$speciesId <- Iris[,1]

sepalData

#A data frame called petalData is initialized as follows
petalData <- Iris[,c(2,3)]

#A primary key is added to data frame sepalData by using seq() function
petalData$petalId <- seq(151,300,by=1)
```

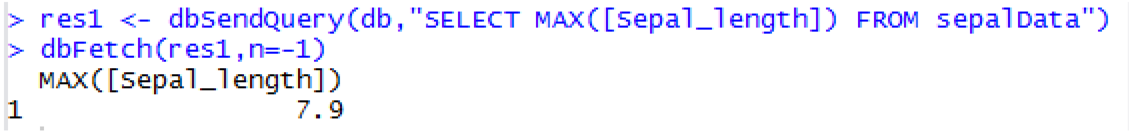


```
#since data frames sepalData and species are related,
#I have added primary key of species table as foreign key to sepalData
petalData$speciesId <- Iris[,1]

petalData

library(RSQLite)

#Make connection to SQLite
db <- dbConnect(SQLite(), dbname="flower.sqlite")

#Import the 3 data frames made into SQLite database using dbWriteTable command
dbWriteTable(conn = db, name = "species", value = species, row.names = FALSE)
dbWriteTable(conn = db, name = "sepalData", value = sepalData, row.names = FALSE)
dbWriteTable(conn = db, name = "petalData", value = petalData, row.names = FALSE)

#Show all tables in the database
dbListTables(db)

#List columns in the table "petalData"
dbListFields(db, "petalData")

res <- dbSendQuery(db,"SELECT MAX (petalID) FROM petalData")
dbFetch(res,n=-1)

res1 <- dbSendQuery(db,"SELECT MAX([Sepal_length]) FROM sepalData")
dbFetch(res1,n=-1)

res2 <- dbSendQuery(db,"SELECT DISTINCT[Petal_width] FROM petalData")
dbFetch(res2,n=-1)

res3 <- dbSendQuery(db, "SELECT [Petal_width] FROM petalData WHERE [speciesId] = 1")
dbFetch(res3,n=-1)
```

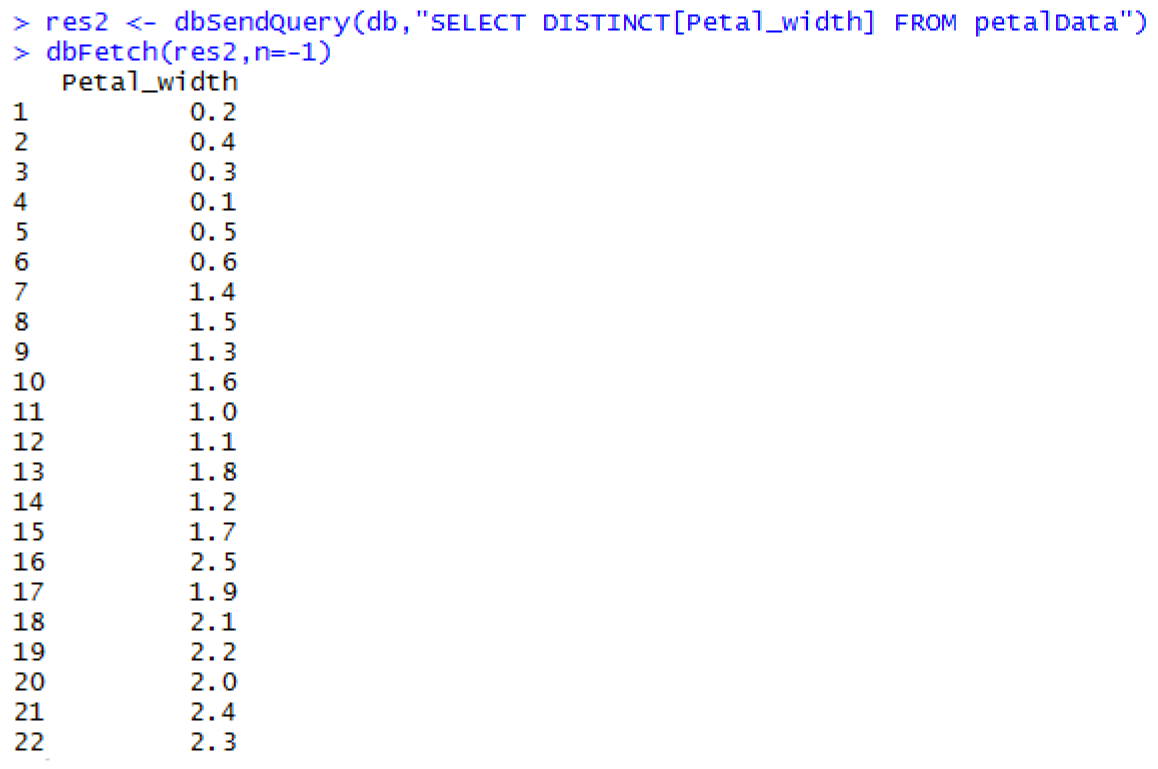
Here are the results of some queries:



```
> res <- dbSendQuery(db,"SELECT MAX (petalID) FROM petalData")
> dbFetch(res,n=-1)
  MAX (petalID)
1           300
```

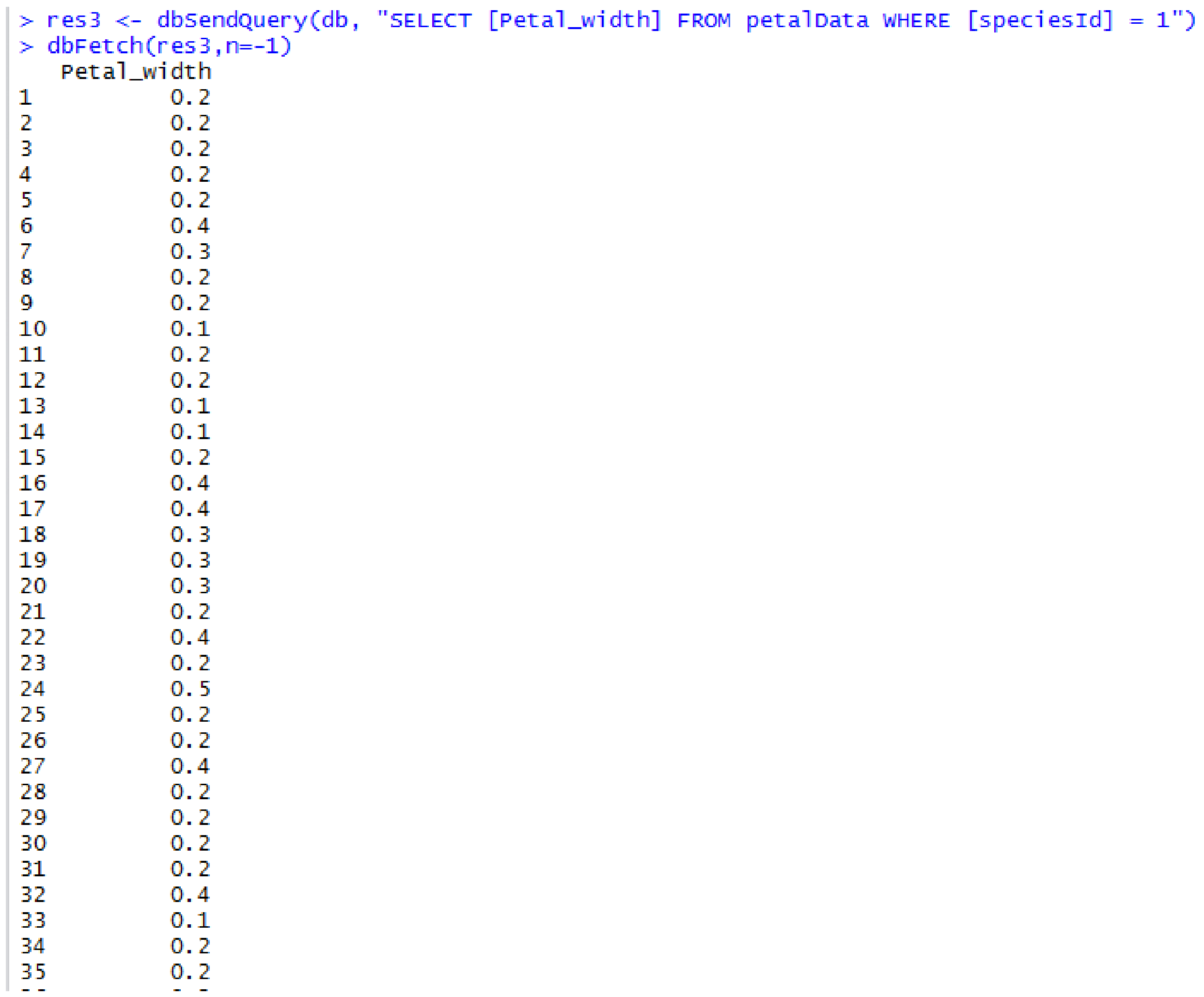


```
> res1 <- dbSendQuery(db,"SELECT MAX([Sepal_length]) FROM sepalData")
> dbFetch(res1,n=-1)
  MAX([Sepal_length])
1                 7.9
```

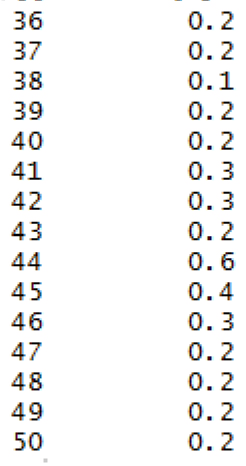


```
> res2 <- dbSendQuery(db,"SELECT DISTINCT[Petal_width] FROM petalData")
> dbFetch(res2,n=-1)
   Petal_width
1          0.2
2          0.4
3          0.3
4          0.1
5          0.5
6          0.6
7          1.4
8          1.5
9          1.3
10         1.6
11         1.0
12         1.1
13         1.8
14         1.2
15         1.7
16         2.5
17         1.9
18         2.1
19         2.2
20         2.0
21         2.4
22         2.3
```

Here is the output of res3. I have shown it in 2 snippets since it was not fitting in a single snippet.



```
> res3 <- dbSendQuery(db, "SELECT [Petal_width] FROM petalData WHERE [speciesId] = 1")
> dbFetch(res3,n=-1)
   Petal_width
1          0.2
2          0.2
3          0.2
4          0.2
5          0.2
6          0.4
7          0.3
8          0.2
9          0.2
10         0.1
11         0.2
12         0.2
13         0.1
14         0.1
15         0.2
16         0.4
17         0.4
18         0.3
19         0.3
20         0.3
21         0.2
22         0.4
23         0.2
24         0.5
25         0.2
26         0.2
27         0.4
28         0.2
29         0.2
30         0.2
31         0.2
32         0.4
33         0.1
34         0.2
35         0.2
```



```
36         0.2
37         0.2
38         0.1
39         0.2
40         0.2
41         0.3
42         0.3
43         0.2
44         0.6
45         0.4
46         0.3
47         0.2
48         0.2
49         0.2
50         0.2
```

## References

https://en.wikipedia.org/wiki/SQL
http://www.1keydata.com/sql/sqlorderby.html

# Go language

This is a programming language, developed at Google and is also popularly known as golang. The syntax is similar to that of language C. Its main features are garbage collection, type safety and dynamic typing capabilities.

## Basics of Go language

### Variables

A variable's type determines which values a variable can hold. The basic types of a variable are as follows:

bool, string, int, int8, int16, int32 (also known as rune), int64, uint, uint8 (also known as byte), uint16, uint32, uint64, uintptr, float 32, float64, complex64, complex128. Other than these types there are several other types which are array, slice, struct, function, pointer, interface, map and channel types.

The var statement is used to declare variables and the type is declared next to the variable's name. The syntax is as follows:

```
var (
           name string
            age int
          city string
     )
```

OR

```
var (
            name,city string
            age int
     )
```

Variables can be declared one by one by preceding each of them with var, instead of grouping them as shown in above example.

Initializers can be used to set the values to variables as follows:

```
var (
      name string = "Anderson"
   age int = 35
   city string = "Boston"
    )
```

In the presence of an initializer, type can be omitted. The variable will take the initializer's type. This is called 'inferred typing'.

```
var (
     name = "Anderson"
      age = 35
     city = "Boston"
    )
```

To initialize them on the same line following is the syntax:

var ( name, age, city = "Anderson", 35, "Boston")

Inside a function, the := short assignment statement can be used in place of var declaration having implicit type. Outside a function the := assignment is not available so the types should be mentioned.

```
package main

import "fmt"

func main() {
      name, city := "Anderson", "Boston"
      age := 35
      fmt.printf( " %s is of age %d lives in %s" , name, age, city)
          }
```

This will return - Anderson is of age 35 lives in Boston.

## Constants

These are of types rune, integer, floating point, complex and string. Rune, integer, complex and floating point constants are together called as numeric constants. These are not declared by the := syntax. An untyped constant takes the type needed by its context.

Following is a code with constants:

```
package main

import "fmt"

const Pi = 3.14

func main() {

          const one = "Everyone"

         fmt.Println("Hello", one)

         fmt.Println("Happy", Pi, "Day")


        const Truth = true

        fmt.Println("Laws of physics are proven", Truth)

          }
```

The code returns:

```
Hello Everyone

Happy 3.14 Day

Laws of physics are proven true
```

Packages and imports

Every Go program has packages. Program starts running in the package 'main'. Other packages can be imported by using 'import' command.

```
import (

     "fmt"
```

```
    "math/rand"
  )
```

After importing the package, we can refer to the name it exports. The name to be exported should start with a capital letter.

An example code is as follows:

```
package main

import (
        "fmt"
        "math"
)


func main() {
        fmt.Println(math.Pi)
}
```

The answer to the above code is 3.14

## Functions

A function can take zero or more arguments. A code showing function usage is as follows:

```
package main

import "fmt"

func add(x, y int) int {
        return x + y
}


func main() {
        fmt.Println(add(315,426))
}
```

The result returns the addition which is 741.

A function can return multiple number of results. If a return statement is displayed without arguments then it returns current values of the result and are called as 'naked' returns. These should only be used in case of short functions because, they can hamper the readability in case of longer functions.

## More types (of variables)

Apart from basic types, there are some additional ones as follows:

Pointers

Like 'C' language, Go also has pointers which hold memory address of a variable.

Following code illustrates use of pointers:

```
package main

import "fmt"

func main() {

        i := 200


        p := &i         // point to i

        fmt.Println(*p) // read i through the pointer

        *p = 500        // change the value of i through the pointer

        fmt.Println(i)  // see the new value of i

            }
```

The result is as follows:

```
200

500
```

Struct

A struct is a collection of fields or properties. An example code is as follows:

```
package main

import "fmt"

type Disney struct {

        X int

        Y int

}


func main() {

        fmt.Println(Disney{5,8})

}
```

Result: {5 8}

Arrays

Array of type n[t] is an array having n values of type t. Indexes are hold in [].

Example code is as follows:

```
package main


import "fmt"


func main() {

        var a [3]string

        a[0] = "Hello"

        a[1] = "World"

        a[2] = "Everyone"
        fmt.Println(a[0], a[1], a[2])

        fmt.Println(a)

}
```

Result:

```
Hello World Everyone

[Hello World Everyone]
```

Slices:

They point to an array and also include a length. []T is a slice with elements that are of type T.

Example code is as follows:

```
package main

import "fmt"

func main() {

      letters := []string{"u", "v", "w", "x", "y", "z"}

      fmt.Println(letters)

            }
```

Result:

```
[u,v,w,x,y,z]
```

Slices can be created with 'make' function, slices can be sliced again to create new slices. New elements can be added to a slice using append function.

Map

It is similar to 'dictionary' in Python or 'hash' in Perl. It maps keys to values.

An example code is as follows:

```
package main
import "fmt"
func main(){
            zips := map[string]int{
              "Woburn" : 01801,
              "Burlington" : 01803
              "Waltham" : 02451
             }
        fmt.printf("%#v", zips)
}
```

Result:

```
map[string]int{"Woburn":01801, "Burlington":01803, "Waltham":02451}
```

## Flow control statements

For

For loop is the only looping construct in Go, somewhat similar syntax to "C".

```
package main

import "fmt"

func main() {
           sum := 0
           for i := 0; i < 5; i++ {
```

```
		sum += i
	}
	fmt.Println(sum)
}
```

Result: 10

For can also be used as "while" in Go.

Example code :

```
package main

import "fmt"

func main() {
	sum := 1
	for sum < 100 {
		sum += sum
	}
	fmt.Println(sum)
}
```

Result: 128

if and if else

It is similar to "C" and "Java"

Example code:

```
package main
```

```
import "fmt"

func main() {

        if 5%2 ==0{

fmt.Println ("5 is even")

         }else{

fmt.Println ("5 is odd")

}

}
```

Result: 5 is odd

Switch case statement

When the selection has to be made on based on the value of a variable, switch case should be used.

Example code:

```
package main

import "fmt"

func main() {
    height := 5

    // Use switch on the height variable.
    switch {
    case height <= 4:
              fmt.Println("Short")
    case height <= 5:
              fmt.Println ("Normal")
    case height > 5:
              fmt.Println("Tall")
    }
}
```

Result: Normal

Defer statement

A defer statement in Go defers the execution of a function until the surrounding function is returned. It is generally used to simplify functions performing clean-up actions.

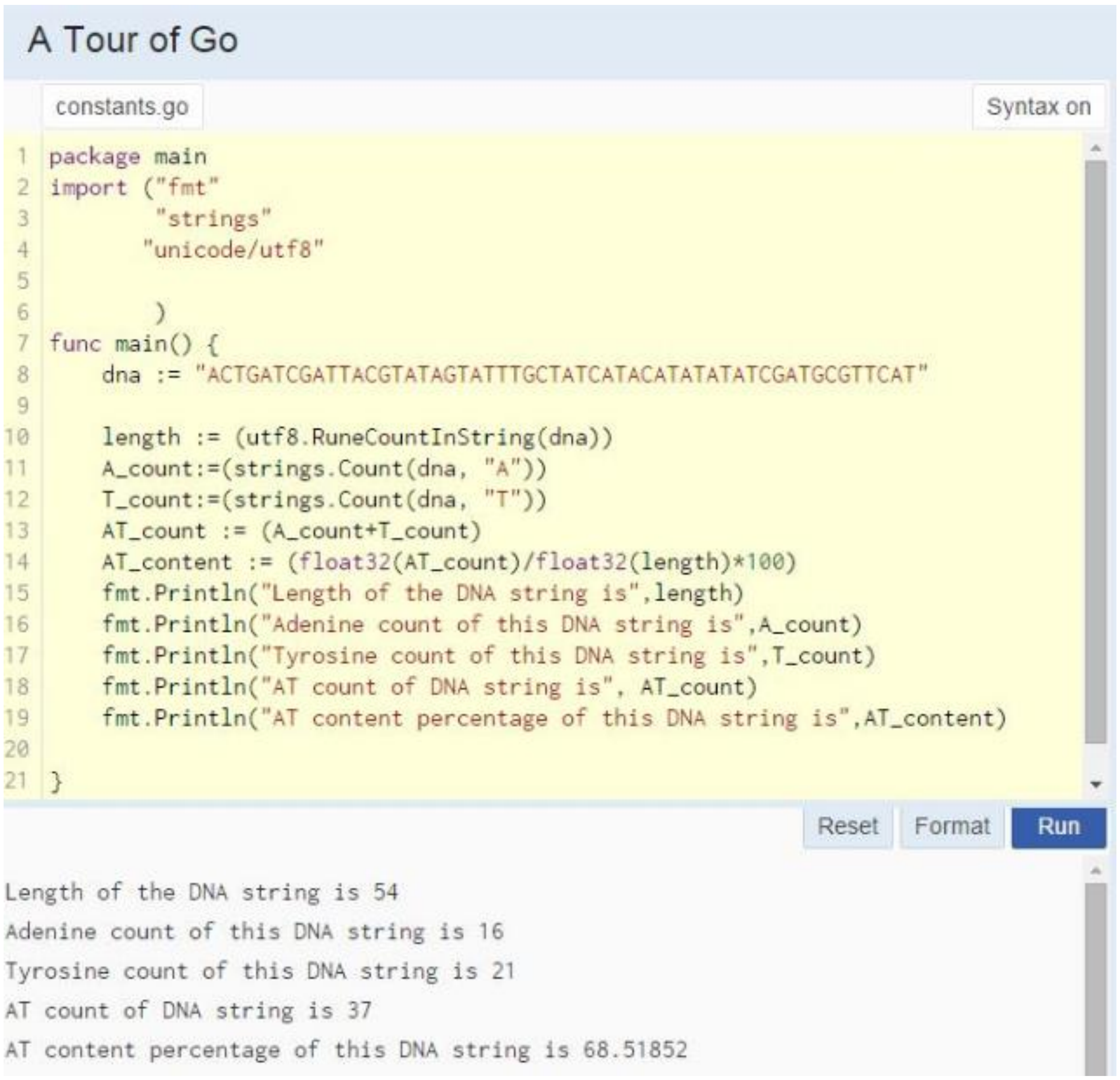
Example code:

```
package main

import "fmt"

func main() {
	defer fmt.Println("world")

	fmt.Println("hello")
}
```

```
hello

world
```

## Code in Go

Following is the code in Go. Please note that I have written the same code in Python.



```go
package main
import ("fmt"
         "strings"
        "unicode/utf8"

         )
func main() {
    dna := "ACTGATCGATTACGTATAGTATTTGCTATCATACATATATATCGATGCGTTCAT"

    length := (utf8.RuneCountInString(dna))
    A_count:=(strings.Count(dna, "A"))
    T_count:=(strings.Count(dna, "T"))
    AT_count := (A_count+T_count)
    AT_content := (float32(AT_count)/float32(length)*100)
    fmt.Println("Length of the DNA string is",length)
    fmt.Println("Adenine count of this DNA string is",A_count)
    fmt.Println("Tyrosine count of this DNA string is",T_count)
    fmt.Println("AT count of DNA string is", AT_count)
    fmt.Println("AT content percentage of this DNA string is",AT_content)

}
```

```
Length of the DNA string is 54
Adenine count of this DNA string is 16
Tyrosine count of this DNA string is 21
AT count of DNA string is 37
AT content percentage of this DNA string is 68.51852
```

## References

https://tour.golang.org
http://www.golangbootcamp.com/book/
http://blog.golang.org/defer-panic-and-recover
https://gobyexample.com
http://www.dotnetperls.com/switch-go

# Python

## Basics of Python

### Variable

In python variable is declared as follows:

```
x = 5

y = "AGTTC"
```

### Change a case

Sometimes it is necessary to change case of a string.  A method can be used to do this. A method is like a function except that it is not build in Python language and belongs to particular type. An example is as follows:

```
dna = "AGTCTA"

print (dna.lower())
```

Result:

```
agtcta
```

### Slicing in Python

This allows one to get the substring from a long string.

Example code:

```
my_dna = "AATCGTACATGCGGTTCATG"

print (my_dna[3:5])
```

Result: CGT

```
print (my_dna[0:4])
```

Result: AATCG

Note that the numbering starts at 0.

## Lists

To hold many pieces of information a list can be used. An example of list is as follows:

```
fruits = ["banana", "mango", "peach", "plum"]
```

To get a particular element out of a list following command can be used:

```
print (fruits[0])
```

Result: banana

Writing a loop

to print all elements of a list, following is the command:

```
for fruit in fruits:
        print (fruit+"\n")
```

```
Result: banana
        mango
      peach
      plum
```

Splitting a string to make a list

```
fruits = "banana, mango, peach, plum"
fruitsName  = fruits.split(",")
print (str(fruitsName))
```

Result:

```
[ "banana","mango","peach","plum"]
```

## Function

It starts with 'def' (short for define), following is the name and then names of argument variables in parentheses. After that a colon is necessary. Following an example of function which counts GC content os a given string of DNA:

```
def get_gc_content(dna):
        length = len(dna)

        g_count = dna.upper().count('G')

        c_count = dna.upper().count('C")

        gc_content = (g_count +c_count)/length

        return (gc_content)
```

To call the function, following is the code:

```
print (get_gc_content("ATGGTCACAATGCACTGACC"))
```

Result: 0.5

Assert

Python's build-in tool 'assert' can be used to test the output of a function using an input which answer is already known.

For example, for the above function, following statement can be used as an assertion.

```
assert get_gc_content("ATGC") ==0.5
```

## Flow Control statements

if statement

In the given if statement, the body of if statement is executed if the condition is true. Following is the syntax of an if statement:

```
if  expressionLevel>95:

        print ("The gene is highly expressed")
```

else statement

It tells what action to perform or what decision should be made if the "if" statement is not executed. For example:

```
if expressionLevel>95:

        print ("The gene is highly expressed")

else:
```

print ("The gene is lowly expressed")

elif statement

this can be used when there are more than two possibilities.

while loop

This is run until some condition is met. An example is as follows:

count = 0

while count<10:

print (count)

count = count+1

## Dictionaries

These are used to store key-value pairs. The syntax to  create a dictionary is as follows:

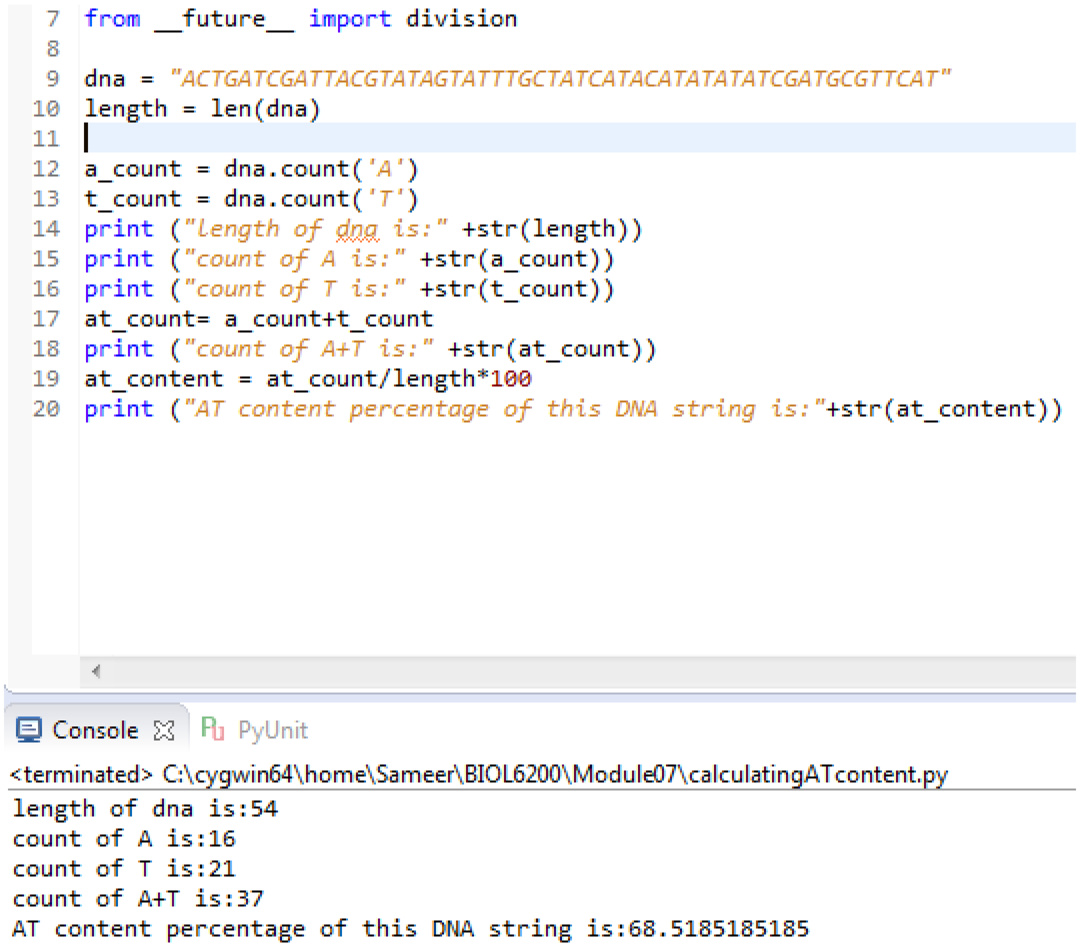
codons = {'I' : 'ATT', 'L' : 'CTT', 'V' ; 'GTT', 'F' : 'TTT', 'M' : 'ATG'}

To access an element from the dictionary, following is the command:

print (codons['M'])

## Code in Python

Following is the code in python. The file calculatingATcontent.py has the following code.



```
from __future__ import division

dna = "ACTGATCGATTACGTATAGTATTTGCTATCATACATATATATCGATGCGTTCAT"
length = len(dna)

a_count = dna.count('A')
t_count = dna.count('T')
print ("length of dna is:" +str(length))
print ("count of A is:" +str(a_count))
print ("count of T is:" +str(t_count))
at_count= a_count+t_count
print ("count of A+T is:" +str(at_count))
at_content = at_count/length*100
print ("AT content percentage of this DNA string is:"+str(at_content))
```

```
<terminated> C:\cygwin64\home\Sameer\BIOL6200\Module07\calculatingATcontent.py
length of dna is:54
count of A is:16
count of T is:21
count of A+T is:37
AT content percentage of this DNA string is:68.5185185185
```

## References

Python for Biologists by Dr. Martin Jones

## Kafka

It is an open source message broker (i.e. intermediary program module which translates a message from the formal messaging protocol of the sender to that of the receiver) project developed by Apache software foundation. It is written in scala. It was originally developed at LinkedIn and then was open sourced in 2011. It is useful for handling real time data feeds.

### Basics of Kafka

Kafka works as a messaging system. Messages are byte arrays. It records messages to a disk and allow subscribers to read and apply the changes to their own stores. The feeds of messages are maintained in categories called 'topics'. The processes that publish messages to kafka are called topic 'producers'. Processes which subscribe to topic are called consumers. Kafka is run as a cluster. Each cluster has one or more servers, each of which is called as a broker.
Clients and servers communicate via a simple, high performance TCP protocol (Transmission Control Protocol).

### Coding examples in Kafka(scala)

The following two coding examples have been reproduced from
http://www.journaldev.com/7905/scala-loop-control-statements-while-do-while-for-loops

While loop example:

```
objectStud{
    def main(args:Array[string]{
var sid = 5;
while (sid < 15) {
 println("Student Id is:" +sid);
 sid = sid + 1;
 }
}
}
```

Answer:

```
Student Id is:5
Student Id is:6
Student Id is:7
Student Id is:8
Student Id is:9
Student Id is:10
Student Id is:11
Student Id is:12
Student Id is:13
Student Id is:14
```

For loop example:

```
object Student {
def main(args:Array[String]) {
   var sid = 0
for (sid <- 6 to 12){
   println("Student Id is :"+sid)
}
}
}
```

Answer:

```
Student Id is:6
Student Id is:7
Student Id is:8
Student Id is:9
Student Id is:10
```

## References

https://en.wikipedia.org/wiki/Apache_Kafka
http://kafka.apache.org/documentation.html#introduction
http://blog.cloudera.com/blog/2014/09/apache-kafka-for-beginners/
http://www.journaldev.com/7905/scala-loop-control-statements-while-do-while-for-loops

## Comparisons of different languages

### R

#### Advantages
1. It is very useful in statistical analysis and dealing with big data.

2. It has numerous graphical capabilities.

3. It offers different packages by which it is possible to use various database management systems through R. The packages are available in range of fields like economics, data mining, spatial analysis, bio-informatics.

4. It facilitates data import in various file formats.

#### Disadvantages
1. It has a steep learning curve. It is not very easy to use for novice.

2. R can quickly consume available memory and hence can pose restrictions in data mining.

Personally I found that data types of R are very easy to understand and comprehend. It is useful in data cleaning and data manipulation before applying that data in various database management systems. So it can be considered as a good front-end analysis tool which can then be coupled with other data management tools. However, for me string processing through regex was difficult. Also handling multiple time formats in R is tricky.

#### References(R/comparison)
http://analyticstrainings.com/?p=101

### Matlab

#### Advantages
1. Its basic data structure is matrix. So it is easy to perform mathematical operations on it. Programs can be easily written and modified and can be debugged with matlab debugger.

2. It performs vectorized operations. So compared to other languages, instead of using for loop or while loop manipulations of arrays can be performed in a simple commands.

3. Graphical output is very optimized for interaction. It is easy to plot a data, add colors, change size or scales, add legends using variety of commands matlab offers.

4.It has number of toolboxes and apps that are application in various fields like mathematics, statistics, signal processing and communication, image processing, biology, computational fianance etc.

### Disadvantages

1. It consumes large amount of memory.

2. Since it is an interpreted language, it can execute more slowly than compiled languages.

I found that matlab was very easy when dealing with numerical data. The commands are very intuitive. However dealing with strings and cell arrays, was somewhat difficult.

### References(matlab/comparison)

http://www.matlabprojecthelp.com/about-matlab/

http://www.mathworks.com/products/

http://www.yorku.ca/jdc/Matlab/Lesson1.htm

## MongoDB

### Advantages

1. The strength lies in its ability to handle huge amounts of data by replication and horizontal scaling.

2. It is very flexible data model. It is not necessary to conform to a schema hence nesting of values is easy.

3. It is easy to use. It has similarity with SQL queries. So users of SQL find it easy to learn language of mangoDB.

### Disadvantages

1. It encourages denormalization of schemas. So a single typo also can cost a lot.

2. Since it is focussed on large datasets, it works best in large clusters, which can require some effort to design and manage.

Since I had worked with SQL, I found it easy to work with queries in mongoDB. It does not need preprocessing. Unlike with SQLite, I first created tables from the original table and then stored those in database, in mongoDB, I just stored the original table and then retrieved data from it. However, I found it difficult to visualize the data easily and so found it little difficult to build queries.

### Reference(comparison/mongoDB)

Seven databases in seven weeks - Eric Redmond and Jim R. Wilson.

## SQL

### Advantages

1. Queries are useful to retrieve large amount of data from a database quickly.

2. Well defined standards exist.

3. No coding required.

4. It is interactive language so easy to communicate with the database.

### Disadvantages

1. It suffers from relatively poor performance scaling.

2. Development is not community driven so has lagged.

3. Developers may find some of its limitations to be frustating.

I found that the language is easy to grasp, at least for basic queries. However for queries that involve more than one tables, it is difficult to write and execute queries.

### References(SQL/comparison)

http://www.cs.iit.edu/~cs561/cs425/VenkatashSQLIntro/Advantages%20&%20Disadvantages.html

https://www.datarealm.com/blog/five-advantages-disadvantages-of-mysql/

## Go

### Advantages

1. It compiles very easily.

2. Go has garbage collection.

3. It has fairly simple syntax.

4. It provides fundamental support for concurrent execution and communication.

### Disadvantages

1. It is still an experimental language, subject to change.

2. It is not very useful with windows yet.

I don't have much experience with Go but I found that it is very easy to grasp its syntax. It really is a simple language. Golang.org offers online facility to write and run a code which is very handy and useful in learning.

### References(Go/comparison)

http://stackoverflow.com/questions/2198529/what-are-the-advantages-and-disadvantages-of-go-programming-language

http://www.quora.com/What-is-the-advantage-of-Googles-Golang-over-Java-and-Scala

https://golang.org/doc/faq

## Python

### Advantages

1. It is easy to learn.

2. It offers great string processing capability, which is useful in biological data manipulation.

3. It is dynamically typed language, providing a good syntactic sugar to ease programming efforts.

4. It offers indentation so it is not necessary to position brackets which makes it easy to code programs.

### Disadvantages

1. Absence of a commercial support point, even for an Open Source project (though this situation is changing)

2. Lack of true multiprocessor support.

For me, handling a biological data involving strings is much easier with python than with R. It is easy to learn. However I found it difficult to write object oriented code in python.

### References(Python/comparison)

http://www.techrepublic.com/article/python-in-the-enterprise-pros-and-cons/

http://www.quora.com/What-are-the-advantages-and-disadvantages-of-Python-compared-to-C++

Python for Biologists by Martin Jones

## Kafka

### Advantages

1. It is a good tool for anyone doing machine learning at large scale.

2. It is expressive and also capable of building robust systems.

3. Good for rapid and real time analysis.

### Disadvantage

It is too fast. Operating in real time lends itself to errors and occasionally kafka misses things.

I have very little experience with kafka. I think it is very powerful messaging tool. I found it difficult to learn basic concepts in kafka and syntax is also difficult to grasp.

### Reference(comparison/Kafka)

http://www.fastcompany.com/3030716/the-9-best-languages-for-crunching-data

## Comments

Based on all the feedback on the discussion forum, I have used same data file in R, matlab, mongoDB and SQL to show data collection, storage and retrieval. Also I wrote the same program in python and Go to show the differences in syntax.

To answer the question whether a particular language or tool is used because of tradition or because of its capabilities, I would elaborate this with the example of matlab and python. Matlab is mathematical language and due to its capabilities it is used widely where numerical data is involved. In case of python, because of its string processing capabilities it is popular in fields like bioinformatics.

To answer a question, if you had to learn only one language which one would that be - I would say it would be definitely R because it is much more versatile and allows interfacing with several other tools like relational and non relational database management systems through its packages.

## List of Attachments

1. fisherIris.R (For R)
2. fisherIris.m (For Matlab)
3. submission_TermProject_mongoDB_Iris.R (For MongoDB)
4. submission_TermProject_mongoDB_FLinsurance.R (For MongoDB)
5. submission_TermProject_SQL.R (For SQL)
6. calculatingATcontent.py (For Python)