Dataset Exploration

Yue Xiong, LMU, yue.xiong@stat.uni-muenchen.de

Chapter 02 Dataset Exploration and Visualization

Introduction

In an ideal world, we will have all of the data we want with all of the desirable properties (no missing values, no errors, standard formats, and so on). However, that is hardly ever true - and we have to work with using our datasets to answer questions of interest as intelligently as possible.

In this r markdown file, we will explore our datasets to answer some questions of interest.

Learning Objectives

This notebook will give you the opportunity to spend some hands-on time with the data.

This notebook will take you around the different ways you can analyze your data. This involves looking at basic metrics in the larger dataset, taking a random sample, creating derived variables, making sense of the missing values, and so on.

This will be done using both `dbplyr`, `dplyr`, `RSQLite` packages in R. The `RSQLite` Python package will give you the opportunity to interact with the database using SQL to pull data into R. Some additional manipulations will be handled by the `tbl()` function in R (by converting your datasets into dataframes).

This notebook will provide an introduction and examples for:

- How to create new tables from the larger tables in database (sometimes called the "analytical frame")
- How to explore different variables of interest
- How to explore aggregate metrics
- How to handle missing values
- How to join newly created tables

Methods

We will be using the `RSQLite` R package to access tables in our database..

To read the results of our queries, we will be using the `tbl()` function provided by the `dplyr` R package, which has the ability to read tabular data from SQL queries into a pandas DataFrame object. For processing with dataframes, we will use various commands to:

- Create statistical summaries
- Create subsets of the data

Within SQL, we will use various queries to:

- select data subsets

- Sum over groups
- create new tables
- Count distinct values of desired variables
- Order data by chosen variables

R Setup

In R, we first use `install.packages()` to install the required packages and then we use `library()` to load the corresponding package. Among the most famous R packages:

- `dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation problems. It works with local dataframes.

- `dbplyr` is similar to `dplyr`, but can deal with remote database tables using exactly the same R code.

- `RSQLite` embeds the 'SQLite' database engine in R and provides an interface compliant with the 'DBI' package.

```
# install.packages(c("dbplyr", "RSQLite", "DBI"))  # install the required packages
library('dbplyr')  # load the corresponding libraries
```

```
## Warning: package 'dbplyr' was built under R version 4.0.5
```

```
library('dplyr')
```

```
## Warning: package 'dplyr' was built under R version 4.0.5
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:dbplyr':
##
##     ident, sql
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library('RSQLite')
```

```
## Warning: package 'RSQLite' was built under R version 4.0.5
```

By prefixing `?`, detailed introduction on the according package will be provided.

```
?RSQLite
```

```
## starting httpd help server ... done
```

It is always useful to set the working directory before basic data analyzing.

```
setwd("F:/hiwi_work_notebook/bdss-notebooks/R_Notebooks")
```

Load the Data

We can execute SQL queries using R to create connections to the corresponding SQL databases.

And the `RSQLite` package, in this case, has made things much easier.

Establish a Connection to the Database

Firstly, we need to specify the database path and then `DBI::dbConnect()` is utilized to connect to this provided database.

```
database_path = "./data_raw/portal_mammals.sqlite"
mammals = DBI::dbConnect(SQLite(), database_path)
```

Furthermore, the `src_dbi()` function can be used to check the tables stored in this database.

```
src_dbi(mammals)
```

```
## src:  sqlite 3.36.0 [F:\hiwi_work_notebook\bdss-notebooks\R_Notebooks\data_raw\portal_mammals.sqlite
## tbls: plots, species, surveys
```

As it shows, there are three tables in this database, i.e., plots, species, surveys.

Formulate Data Query

Depending on what data we are interested in, we can use different queries to pull different data. In this example, we will pull all the content of the offenders data.

Pull Data from the Database

After formulating the queries, we will be accessing the data and use the `tbl()` function to display the results in a table manner.

Analysis: Using R and SQL

What are the characteristics of the survey

Before we go any further, let's take a look at some of the data that we're working with.

portal-mammals Data

Just like a spreadsheet with multiple worksheets, a SQLite database can contain multiple tables. In this case three of them are listed in the tbls row in the output above:

- plots
- species
- surveys

Identifying Missing Values

We might be concerned about missing values in our data. Let's take a look at some inmate data to show an example of how we might find them.

```
query = "select * from surveys limit 20"
surveys = tbl(mammals, sql(query))
```

```
head(surveys)
```

```
## # Source:   lazy query [?? x 9]
## # Database: sqlite 3.36.0
## #   [F:\hiwi_work_notebook\bdss-notebooks\R_Notebooks\data_raw\portal_mammals.sqlite]
##   record_id month   day  year plot_id species_id sex   hindfoot_length weight
##       <int> <int> <int> <int>   <int> <chr>      <chr>           <int>  <int>
## 1         1     7    16  1977       2 NL         M                  32     NA
## 2         2     7    16  1977       3 NL         M                  33     NA
## 3         3     7    16  1977       2 DM         F                  37     NA
## 4         4     7    16  1977       7 DM         M                  36     NA
## 5         5     7    16  1977       3 DM         M                  35     NA
## 6         6     7    16  1977       1 PF         M                  14     NA
```

Here, we use the `head()` method to look at the top few rows of the surveys data. As you can see, we have lots of information about the mammals, such as record_id, plot_id, species_id, sex, etc. Let's see all of the types of variables that we have in this table using `colnames()`.

```
colnames(surveys)
```

```
## [1] "record_id"       "month"           "day"             "year"
## [5] "plot_id"         "species_id"      "sex"             "hindfoot_length"
## [9] "weight"
```

Identifying Missing Values

We might be concerned about missing values in our data. Let's take a look at some surveys data using `is.na()` to show an example of how we might find them.

```
is.na(surveys$sex)
```

```
## logical(0)
```

Since the `ncdoc.db` uploaded is empty, in this case similar data queries cannot be used for data exploration.

Date Variable

Unfortunately, the database I used here does not have the date attribute. I will leave it as it is until we get access to the original `ncdoc.db` database.

For reference, the query example is shown below:

```
SELECT *, CAST(strftime("%Y",ACTUAL_SENTENCE_END_DATE) as integer) as release_year
FROM sentences
WHERE release_year >= 1980 AND release_year < 1990
```

Summary Statistics

In this section, we look at aggregate statistics on the data. We'll start by looking at the surveys dataset.

```
qry = "SELECT year, species_id, plot_id FROM surveys"
surveys_new = tbl(mammals, sql(qry))
head(surveys_new)
```

```
## # Source:   lazy query [?? x 3]
## # Database: sqlite 3.36.0
## #   [F:\hiwi_work_notebook\bdss-notebooks\R_Notebooks\data_raw\portal_mammals.sqlite]
##    year species_id plot_id
##   <int> <chr>        <int>
## 1  1977 NL               2
## 2  1977 NL               3
## 3  1977 DM               2
## 4  1977 DM               7
## 5  1977 DM               3
## 6  1977 PF               1
```

In order to get the corresponding dataframe, we can use the `data.frame()` function to change the above `list` to `dataframe`.

```
df = data.frame(surveys_new)
head(df, n=10)
```

```
##    year species_id plot_id
## 1  1977         NL       2
## 2  1977         NL       3
## 3  1977         DM       2
## 4  1977         DM       7
## 5  1977         DM       3
## 6  1977         PF       1
## 7  1977         PE       2
## 8  1977         DM       1
## 9  1977         DM       1
## 10 1977         PF       6
```

```
nrow(df)  # nrow() cannot be used with surveys_new: return NA?
```

```
## [1] 35549
```

We can get simple descriptive statistics using the `summary()` function.

```
summary(df)
```

```
##       year       species_id           plot_id
##  Min.   :1977   Length:35549       Min.   : 1.0
##  1st Qu.:1984   Class :character   1st Qu.: 5.0
##  Median :1990   Mode  :character   Median :11.0
##  Mean   :1990                      Mean   :11.4
##  3rd Qu.:1997                      3rd Qu.:17.0
##  Max.   :2002                      Max.   :24.0
```

Let's find out how many unique inmates there were within the above shown time period.

```
length(unique(df$species_id))
```

```
## [1] 49
```

It indicates 49 species observed from year 1977 to 2002.

Now, let's look at the characteristics of the mammals by year. First, let's look at how many mammals there were in each year.

```
ob_by_year = df %>% group_by(df$year)
ob_by_year
```

```
## # A tibble: 35,549 x 4
## # Groups:   df$year [26]
##     year species_id plot_id `df$year`
##    <int> <chr>        <int>     <int>
##  1  1977 NL               2      1977
##  2  1977 NL               3      1977
##  3  1977 DM               2      1977
##  4  1977 DM               7      1977
##  5  1977 DM               3      1977
##  6  1977 PF               1      1977
##  7  1977 PE               2      1977
##  8  1977 DM               1      1977
##  9  1977 DM               1      1977
## 10  1977 PF               6      1977
## # ... with 35,539 more rows
```

```
count(ob_by_year)
```

```
## # A tibble: 26 x 2
## # Groups:   df$year [26]
##    `df$year`     n
##        <int> <int>
##  1      1977   503
##  2      1978  1048
##  3      1979   719
## 4      1980  1415
## 5      1981  1472
## 6      1982  1978
## 7      1983  1673
## 8      1984   981
## 9      1985  1438
## 10     1986   942
## # ... with 16 more rows
```

Note that we first create an aggregated object using the `group_by()` method. Then, we use it to perform certain calculations on each of the groups. Since we grouped by `year`, we are able to obtain various statistics of other variables within each year.

Now, let's look at how many unique mammal there are within each year.

```
unique(ob_by_year)
```

```
## # A tibble: 4,741 x 4
## # Groups:   df$year [26]
##     year species_id plot_id ‘df$year‘
##    <int> <chr>        <int>     <int>
##  1  1977 NL               2      1977
##  2  1977 NL               3      1977
##  3  1977 DM               2      1977
##  4  1977 DM               7      1977
##  5  1977 DM               3      1977
##  6  1977 PF               1      1977
##  7  1977 PE               2      1977
##  8  1977 DM               1      1977
##  9  1977 PF               6      1977
## 10  1977 DS               5      1977
## # ... with 4,731 more rows
```

And we can plot the graph from it.

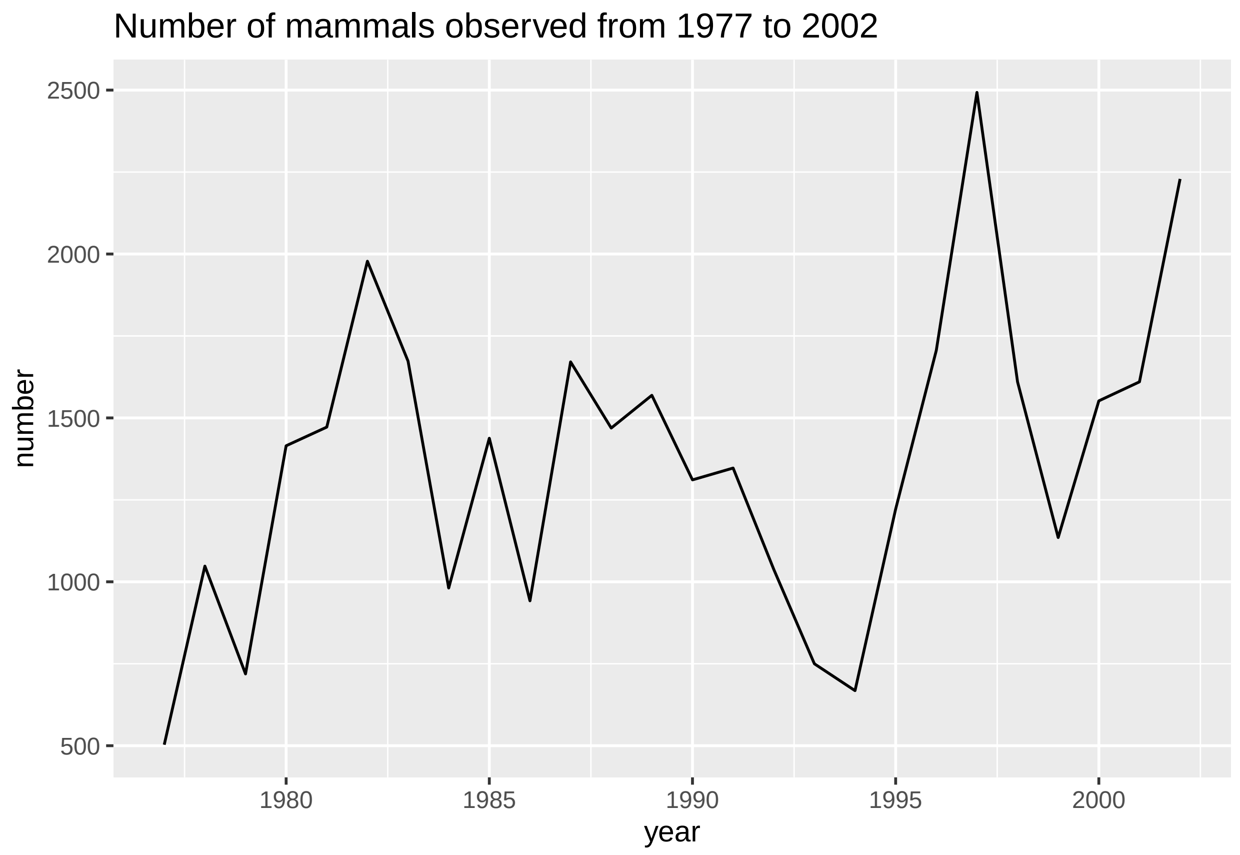
```
# install.packages('ggplot2')
library("ggplot2")
```

```
## Warning: package ’ggplot2’ was built under R version 4.0.5
```

```
plot_data = count(ob_by_year)
colnames(plot_data) = c("year", "number")
plot_data$year
```

```
##  [1] 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
## [16] 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002
```

```
ggplot(data = plot_data, aes(x = year, y = number)) + geom_line() + ggtitle("Number of mammals observed
```



I will stop here as we are lacking information with the `ncdoc.db` database. Further details can be discussed via future meetings.