Introduction to R

Ali Zaidi, Machine Learning and Data Science Education Team

May 17, 2016

Overview of the R Project

Lesson Plan

R U Ready?

- What the R Language for Statistical Computing is
- R's capabilities and it's limitations
- What types of problems you might want to use R with
- How to manage data with open source R
- How to develop models and write functions in R

What is R?

Why should I care?

- R is a successor of the S Language, originated at Bell Labs AT&T
- Based on the Scheme interpreter
- Originally designed by two University of Auckland Professors for their introduction to statistics course

R Philosophy

What R Thou?

R follows the Unix philosophy

Write programs that do one thing and do it well. Write programs to work together.

- R is extensible with more than 9,000 packages available at CRAN (http://crantastic.org/packages)
- R, like it's inspiration, Scheme, is a functional programming language
- R is lazy, and lazy evaluation can be used to interface to other languages
- R is a highly interpreted dynamically typed language, allowing you to mutate variables and analyze datasets quickly, but is significantly slower than low-level, statically typed languages like C or Java
- R has a high memory footprint, and can easily lead to crashes if you aren't careful

Development Environments

Where to Write R Code

- The most popular integrated development environment for R is RStudio
- The RStudio IDE is entirely html/javascript based, so completely cross-platform
- RStudio Server for cloud instances
- Developers of RStudio have also written a plethora of useful R packages
- For Windows machines, we have recently announced the general availability of R Tools for Visual Studio, RTVS
- RTVS will support connectivity to Azure and SQL Server very soon
- RTVS has great debugging support

Quick Tour of Your IDE

Strengths of R

Where R Succeeds

- Expressive
- Open source
- Extendable – nearly 1000 packages with functions to use
- Focused on statistics and machine learning – utilized by academics and practitioners
- Advanced data structures and graphical capabilities
- Large user community, academics and industry
- It is designed by statisticians

Weaknesses of R

Where R Falls Short

- It is designed by statisticians
- Inefficient at element-by-element computations
- May make large demands on system resources, namely memory
- Data capacity limited by memory
- Single-threaded

Some Essential Open Source Packages

- There are over 1,000 R packages to choose from, what do I start with?
- Data Management: `dplyr`, `tidyr`, `data.table`
- Visualization: `ggplot2`, `ggvis`, `htmlwidgets`, `shiny`
- Data Importing: `haven`, `RODBC`
- Other favorites; `magrittr`, `rmarkdown`, `caret`

R Foundations

Command line prompts

Symbol	Meaning
`>`	ready for a new command
`+`	awaiting the completion of an existing command

I'm Lost!

Getting Help with R

- Stack Overflow
- Cross Validated, R
- R Reference Card
- RStudio Cheat Sheets
- R help mailing list and archives
- CRAN Task Views
- Crantastic
- Revolutions Blog
- R-Bloggers

Quick Tour of Things You Need to Know

Data Structures

- R's data structures can be described by their dimensionality, and their type.

	Homogeneous	Heterogeneous
1d	Atomic vector	List
2d	Matrix	Data frame
nd	Array	

Quick Tour of Things You Need to Know

Data Types

- Atomic vectors come in one of four types
- `logical` (boolean). Values: `TRUE` | `FALSE`
- `integer`
- `double` (often called numeric)
- `character`
- Rare types:
- `complex`
- `raw`

Lab 1: R Data Types

Data Manipulation with the dplyr Package

Overview

At the end of this session, you will have learned:

- How to manipulate data quickly with `dplyr` using a very intuitive 'grammar'
- How to use `dplyr` to perform common exploratory and manipulation procedures
- Apply your own custom functions to group manipulations `dplyr` with `mutate()`, `summarise()` and `do()`
- Connect to remote databases to work with larger than memory datasets

Why use dplyr?

The Grammar of Data Manipulation

- R comes with a plethora of base functions for data manipulation
- `dplyr` makes data manipulation easier by providing a few functions for the most common tasks and procedures
- `dplyr` achieves remarkable speed-up gains by using a C++ backend
- `dplyr` has multiple backends for working with data stored in various sources: SQLite, MySQL, bigquery, and more
- `dplyr` was inspired to give data manipulation a simple, cohesive grammar (similar philosophy to `ggplot` - grammar of graphics)
- the recent package `dplyrXdf` brings much of the same functionality of `dplyr` to `XDF` data

Manipulation verbs

filter
select rows based on matching criteria

slice
select rows by number

select
select columns by column names

arrange
reorder rows by column values

mutate
add new variables based on transformations of existing variables

transmute
transform and drop other variables

Aggregation verbs

group_by

identify grouping variables for calculating groupwise summary statistics

count

count the number of records per group

summarise

calculate one or more summary functions per group, returning one row of results per group (or one for the entire dataset)

Viewing Data

- `dplyr` includes a wrapper called `tbl_df` makes df into a 'local df' that improves the printing of dataframes in the console
- if you want to see more of the data you can still coerce to `data.frame`

```
library(dplyr)
bankData <- read.table("data/bank-full.csv",
                       header = T, sep = ";",
                       stringsAsFactors = F)
(bankData <- tbl_df(bankData))
```

```
## Source: local data frame [45,211 x 17]
##
##    age          job  marital education default balance housing loan
## 1   58   management  married  tertiary      no    2143     yes   no
## 2   44   technician   single secondary      no      29     yes   no
## 3   33 entrepreneur  married secondary      no       2     yes  yes
## 4   47  blue-collar  married   unknown      no    1506     yes   no
## 5   33      unknown   single   unknown      no       1      no   no
## 6   35   management  married  tertiary      no     231     yes   no
## 7   28   management   single  tertiary      no     447     yes  yes
## 8   42 entrepreneur divorced  tertiary     yes       2     yes   no
## 9   58      retired  married   primary      no     121     yes   no
## 10  43   technician   single secondary      no     593     yes   no
## .. ...          ...      ...       ...     ...     ...     ...  ...
## Variables not shown: contact (chr), day (int), month (chr), duration
##   (int), campaign (int), pdays (int), previous (int), poutcome (chr), y
##   (chr)
```

Filtering and Reordering Data

Subsetting Data

- `dplyr` makes subsetting by rows very easy
- The `filter` verb takes conditions for filtering rows based on conditions

```
filter(bankData,
       month %in% c("april", "may", "jun"),
       default == "yes")
```

```
## Source: local data frame [351 x 17]
##
##    age          job  marital education default balance housing loan
## 1   42 entrepreneur divorced  tertiary     yes       2     yes   no
## 2   55     services divorced secondary     yes       1     yes   no
## 3   51       admin.   single secondary     yes      -2      no   no
## 4   33   technician  married secondary     yes      72     yes   no
## 5   33  blue-collar   single secondary     yes     -60      no   no
## 6   60      retired  married secondary     yes      15      no   no
## 7   35 entrepreneur  married secondary     yes     204     yes   no
## 8   41  blue-collar   single   primary     yes    -137     yes  yes
## 9   54     services  married secondary     yes       0     yes   no
## 10  52  blue-collar divorced   primary     yes    -183     yes   no
## .. ...          ...      ...       ...     ...     ...     ...  ...
## Variables not shown: contact (chr), day (int), month (chr), duration
##   (int), campaign (int), pdays (int), previous (int), poutcome (chr), y
##   (chr)
```

Exercise

Your turn:

- How many defaults in the dataset?
- How many of the entrepeneurs that defaulted were also divorced?

Solution

```
defaults <- filter(bankData, default == "yes")
nrow(defaults)
```

```
## [1] 815
```

```
brokeEntrepreneurs <- filter(bankData,
                             job == "entrepreneur",
                             marital == "divorced",
                             default == "yes")
nrow(brokeEntrepreneurs)
```

```
## [1] 6
```

Select a set of columns

- You can use the `select()` verb to specify which columns of a dataset you want
- This is similar to the `keep` option in SAS's data step.
- Use a colon `:` to select all the columns between two variables (inclusive)
- Use `contains` to take any columns containing a certain word/phrase/character

Select Example 1

```
select(bankData, age, job, default, balance, housing)
```

```
## Source: local data frame [45,211 x 5]
## 
##    age          job default balance housing
## 1   58   management      no    2143     yes
## 2   44   technician      no      29     yes
## 3   33 entrepreneur      no       2     yes
## 4   47  blue-collar      no    1506     yes
## 5   33      unknown      no       1      no
## 6   35   management      no     231     yes
## 7   28   management      no     447     yes
## 8   42 entrepreneur     yes       2     yes
## 9   58      retired      no     121     yes
## 10  43   technician      no     593     yes
## .. ...          ...     ...     ...     ...
```

Select: Other Options

starts_with(x, ignore.case = FALSE)

name starts with `x`

ends_with(x, ignore.case = FALSE)

name ends with `x`

matches(x, ignore.case = FALSE)

selects all variables whose name matches the regular expression `x`

num_range("V", 1:5, width = 1)

selects all variables (numerically) from `V1` to `V5`.

You can also use a `-` to drop variables.

Reordering Data

- You can reorder your dataset based on conditions using the `arrange()` verb

```
arrange(bankData, balance, default)
```

```
## Source: local data frame [45,211 x 17]
##
##    age           job  marital education default balance housing loan
## 1   26   blue-collar   single secondary     yes   -8019      no  yes
## 2   49    management  married  tertiary     yes   -6847      no  yes
## 3   60    management divorced  tertiary      no   -4057     yes   no
## 4   43    management  married  tertiary     yes   -3372     yes   no
## 5   57 self-employed  married  tertiary     yes   -3313     yes  yes
## 6   39 self-employed  married  tertiary      no   -3058     yes  yes
## 7   40    technician  married  tertiary     yes   -2827     yes  yes
## 8   52    management  married  tertiary      no   -2712     yes  yes
## 9   49   blue-collar   single   primary     yes   -2604     yes   no
## 10  51    management divorced  tertiary      no   -2282     yes  yes
## .. ...           ...      ...       ...     ...     ...     ...  ...
## Variables not shown: contact (chr), day (int), month (chr), duration
##   (int), campaign (int), pdays (int), previous (int), poutcome (chr), y
##   (chr)
```

Exercise

Use `arrange()` to sort on the basis of `y`, `marital`, `job` (in descending order), and `balance`

Solution

```
arrange(bankData, y, marital, desc(job), balance)
```

```
## Source: local data frame [45,211 x 17]
##
##    age     job  marital education default balance housing loan   contact
## 1   26 unknown divorced secondary      no    -295     yes   no  cellular
## 2   47 unknown divorced   primary      no       0      no   no  cellular
## 3   82 unknown divorced   unknown      no       0      no   no telephone
## 4   59 unknown divorced   unknown      no      27      no   no   unknown
## 5   31 unknown divorced  tertiary      no     137      no   no  cellular
## 6   56 unknown divorced   primary      no     558      no   no   unknown
## 7   51 unknown divorced   unknown      no    1649      no   no   unknown
## 8   66 unknown divorced   unknown      no    1993     yes   no  cellular
## 9   56 unknown divorced   unknown      no    2152      no   no   unknown
## 10  53 unknown divorced  tertiary      no    2272     yes   no   unknown
## .. ...     ...      ...       ...     ...     ...     ...  ...       ...
## Variables not shown: day (int), month (chr), duration (int), campaign
##   (int), pdays (int), previous (int), poutcome (chr), y (chr)
```

Summary

filter

Extract subsets of rows. See also `slice()`

select

Extract subsets of columns. See also `rename()`

arrange

Sort your data

Transformations

- The `mutate()` verb can be used to make new columns

```
mutate(bankData,
       DefaultFlag = ifelse(default == 'yes',
                            yes =  1,
                            no = 0)
)
```

```
## Source: local data frame [45,211 x 18]
##
##    age          job  marital education default balance housing loan
## 1   58   management  married  tertiary      no    2143     yes   no
## 2   44   technician   single secondary      no      29     yes   no
## 3   33 entrepreneur  married secondary      no       2     yes  yes
## 4   47  blue-collar  married   unknown      no    1506     yes   no
## 5   33      unknown   single   unknown      no       1      no   no
## 6   35   management  married  tertiary      no     231     yes   no
## 7   28   management   single  tertiary      no     447     yes  yes
## 8   42 entrepreneur divorced  tertiary     yes       2     yes   no
## 9   58      retired  married   primary      no     121     yes   no
## 10  43   technician   single secondary      no     593     yes   no
## .. ...          ...      ...       ...     ...     ...     ...  ...
## Variables not shown: contact (chr), day (int), month (chr), duration
##   (int), campaign (int), pdays (int), previous (int), poutcome (chr), y
##   (chr), DefaultFlag (dbl)
```

Summarise Data by Groups

- The `group_by` verb creates a grouping by a categorical variable
- Functions can be placed inside `summarise` to create summary functions

```
summarise(group_by(bankData, default), Num = n())
```

```
## Source: local data frame [2 x 2]
##
##   default   Num
## 1      no 44396
## 2     yes   815
```

Example group_by 2

```
summarise(group_by(bankData, default), ave_balance = mean(balance))
```

```
## Source: local data frame [2 x 2]
## 
##   default ave_balance
## 1      no   1389.8064
## 2     yes   -137.6245
```

Example group_by 3

```
summarise(group_by(bankData, default),
          ave_balance = mean(balance),
          num = n()
)
```

```
## Source: local data frame [2 x 3]
##
##   default ave_balance   num
## 1      no   1389.8064 44396
## 2     yes   -137.6245   815
```

Chaining/Piping

- A `dplyr` installation includes the `magrittr` package as a dependency
- The `magrittr` package includes a pipe operator that allows you to pass the current dataset to another function
- This makes interpreting a nested sequence of operations much easier to understand

Standard Code

Code is executed inside-out.

```r
arrange(filter(select(bankData, age, job, education, default), default == 'yes'), job, education, age)
```

```
## Source: local data frame [815 x 4]
## 
##    age    job education default
## 1   25 admin. secondary     yes
## 2   26 admin. secondary     yes
## 3   26 admin. secondary     yes
## 4   26 admin. secondary     yes
## 5   26 admin. secondary     yes
## 6   27 admin. secondary     yes
## 7   27 admin. secondary     yes
## 8   27 admin. secondary     yes
## 9   28 admin. secondary     yes
## 10  29 admin. secondary     yes
## .. ...    ...       ...     ...
```

Reformatted

```
arrange(
  filter(
    select(bankData, age, job, education, default),
    default == 'yes'),
  job, education, age)
```

```
## Source: local data frame [815 x 4]
##
##    age    job education default
## 1   25 admin. secondary     yes
## 2   26 admin. secondary     yes
## 3   26 admin. secondary     yes
## 4   26 admin. secondary     yes
## 5   26 admin. secondary     yes
## 6   27 admin. secondary     yes
## 7   27 admin. secondary     yes
## 8   27 admin. secondary     yes
## 9   28 admin. secondary     yes
## 10  29 admin. secondary     yes
## .. ...    ...       ...     ...
```

With Piping

```
bankData %>%
  select(age, job, education, default) %>%
  filter(default == 'yes') %>%
  arrange(job, education, age)
```

```
## Source: local data frame [815 x 4]
##
##    age    job education default
## 1   25 admin. secondary     yes
## 2   26 admin. secondary     yes
## 3   26 admin. secondary     yes
## 4   26 admin. secondary     yes
## 5   26 admin. secondary     yes
## 6   27 admin. secondary     yes
## 7   27 admin. secondary     yes
## 8   27 admin. secondary     yes
## 9   28 admin. secondary     yes
## 10  29 admin. secondary     yes
## .. ...    ...       ...     ...
```

Pipe + group_by()

The pipe operator is very helpful for group by summaries

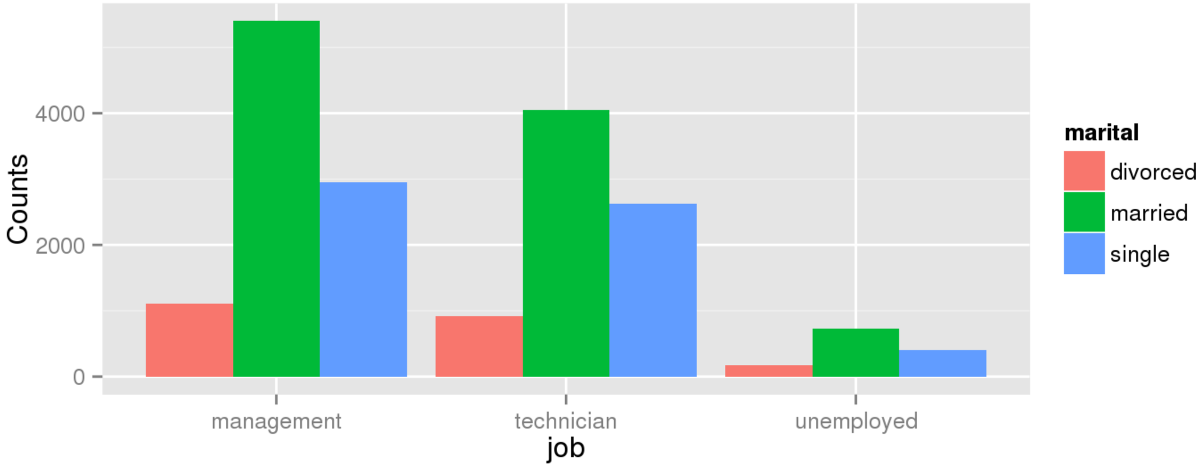
```
bankData %>%
  group_by(job) %>%
  summarise(num = n(),
            ave_balance = mean(balance),
            num_defaults = sum(default == 'yes'),
            default_rate = num_defaults/num)
```

```
## Source: local data frame [12 x 5]
##
##              job  num ave_balance num_defaults default_rate
## 1         admin. 5171   1135.8389           74  0.014310578
## 2    blue-collar 9732   1078.8267          201  0.020653514
## 3   entrepreneur 1487   1521.4701           55  0.036987223
## 4      housemaid 1240   1392.3952           22  0.017741935
## 5     management 9458   1763.6168          164  0.017339818
## 6        retired 2264   1984.2151           26  0.011484099
## 7  self-employed 1579   1647.9709           33  0.020899303
## 8       services 4154    997.0881           75  0.018054887
## 9        student  938   1388.0608            3  0.003198294
## 10    technician 7597   1252.6321          130  0.017112018
## 11    unemployed 1303   1521.7460           30  0.023023791
## 12       unknown  288   1772.3576            2  0.006944444
```

Pipe and Plot

As a reminder, piping can also be used for non-dplyr functions.

```
library(ggplot2)
bankData %>%
  filter(job %in% c("management", "technician", "unemployed")) %>%
  group_by(job, marital) %>%
  summarise(Counts = n()) %>%
  ggplot() +
  geom_bar(aes(x = job, y = Counts, fill = marital),
           stat = 'identity', position = 'dodge')
```



Exercise

Your turn:

- Use the pipe operator to group by job and housing status
- Calculate the counts of observations, and the average and median balance

Solution

```
bankData %>%
  group_by(job, housing) %>%
  summarise(Counts = n(),
            average_balance = mean(as.numeric(balance)),
            median_balance = median(as.numeric(balance)))
```

```
## Source: local data frame [24 x 5]
## Groups: job
##
##              job housing Counts average_balance median_balance
## 1         admin.      no   1989       1378.7677          445.0
## 2         admin.     yes   3182        983.9893          387.0
## 3    blue-collar      no   2684       1341.3308          421.5
## 4    blue-collar     yes   7048        978.8605          373.0
## 5   entrepreneur      no    618       1862.7023          390.0
## 6   entrepreneur     yes    869       1278.7986          339.0
## 7      housemaid      no    842       1491.1876          465.5
## 8      housemaid     yes    398       1183.3920          314.5
## 9     management      no   4780       1913.7525          600.5
## 10    management     yes   4678       1610.2076          547.0
## ..           ...     ...    ...             ...            ...
```

Summary

mutate

Create transformations

summarise

Aggregate

group_by

Group your dataset by levels

distinct

Extract unique values (frequently used with `arrange()`)

Chaining with the `%>%` operator can result in more readable code.

Thanks for Attending!

- Any questions?
- alizaidi@microsoft.com