A (Re)Introduction to R

Analytics Kaizen

February 28, 2018

R

R is an open-source programming language for statistical computing, analysis, and data science.

R
cran.r-project.org

RStudio
www.rstudio.com

R/RStudio Orientation

- Console: Where you run code.
- Source: Create and save R scripts and send code to the Console.

```
# Use comments to explain your code.
2 * (18 - 7)
```

- Environment: A snapshot of what you have loaded.
- Help: Look up documentation.

```
?library
```

RStudio Preferences

- General > Save workspace to .RData on exit: Never.
- Code > Editing > Execution > Ctrl + Enter executes: Multiple consecutive R lines.
- Code > Display > General > Highlight selected line.
- You can also pick a different color palette.
- Make note of the cheatsheets under Help.

RStudio Projects and Importing Data

- We'll want to import data, but R needs to know where to look.
- Computer files are stored in a series of folders (i.e., directories) and each have a file path. For example (on my Mac):

  `/Users/marcdotson/Documents/Analytics`

- You should have a directory for this tutorial. Create an RStudio Project using that directory.
- Whenever you have this project open in RStudio, this directory will be R's working directory (see `getwd()`).
- Now download some data and store it in your tutorial directory: github.com/marcdotson/re-introduction-to-r

R Packages

- An R package is a collection of functions, documentation, and sometimes data.
- There are a number of packages that are part of the base installation of R (look at Packages tab).
- You can download and load other packages from CRAN.

```
# install.packages("<PACKAGE_NAME>")
# library(<PACKAGE_NAME>)
```

- Not all packages are created equal.

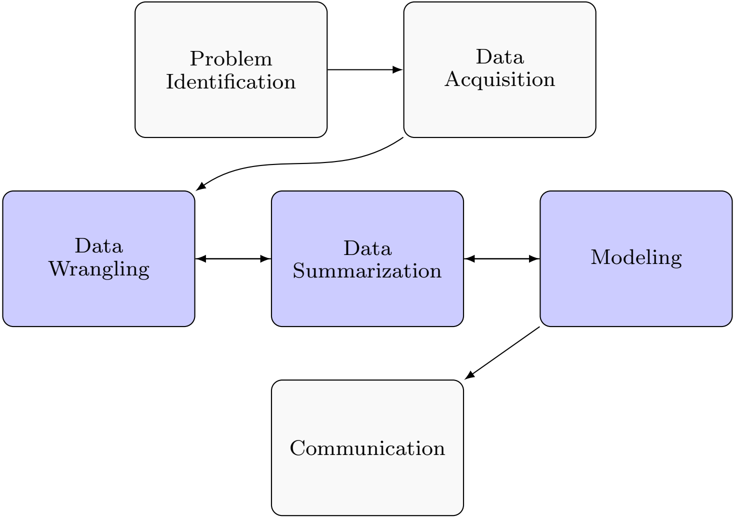
The Tidyverse

- The tidyverse is a collection of R packages that share common philosophies and are designed to work together.
  - readr – importing data
  - dplyr – manipulating data
  - tidyr – cleaning data
  - ggplot2 – visualizing data
- The tidyverse represents the state of the art for data wrangling and summarization in R and serves as the foundation for a growing number of additional packages.

```
install.packages("tidyverse")
library(tidyverse)
```

- For more detail about the tidyverse, visit the website and watch Hadley Wickham explain.

Marketing Analytics Process



Data Frames

- Data frames are the most common data structure in R.

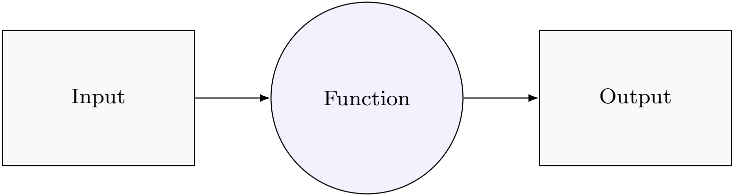
  ```
  store_data <- read_csv("store_data.csv")
  ```

- Note that since we've assigned it a name, `store_data` appears in our Environment.
- We can view any object in its own tab.
- How many observations and variables?
- Each column has a name. What is consistent about how the columns are named?
- Each column is also known as a vector (a data structure all its own) where each vector can be a different data type. What types do you see here?

Data Manipulation

- The heart of data wrangling is data manipulation.
  - Filter observations.
  - Arrange observations.
  - Select variables.
  - Recode or create new variables.
  - Join data frames.
- One of the most-used packages, dplyr provides a consistent grammar of data manipulation with functions (a.k.a., verbs) that mirror SQL.

Functions



- Functions are composed of arguments that tell how the function how to operate.
- Using a function is referred to as a “call” or a “function call.”
- One of the common philosophies of the tidyverse is to `%>%` together functions in consecutive lines of code.
  - Each function has been designed to do one specific thing well.
  - Each function has a data frame as an input and a data frame as an output.
  - While you still want to comment code, it makes it easy to read.
- Don’t forget you have Help (Slides > Documentation > Stake Overflow > Google)

Filtering Observations

- We often want to filter our data by keeping certain observations.

```
store_data %>%
  filter(gender == "Female")

store_data %>%
  filter(store_spend > 100)
```

- How would we filter by `gender == "Female"` and `store_spend >100`?
- Why are we putting quotes around `"Female"` but not `gender`?

Arrange Observations

- Arrange observations to reveal helpful information and check data.

```
store_data %>%
  arrange(store_trans)

store_data %>%
  arrange(desc(store_trans))
```

Selecting Variables

- Sometimes we only care about keeping certain variables, especially when working with a large dataset.

```
store_data %>%
  select(store_spend, age, gender)
```

Recoding/Creating New Variables

- We can also recode existing variables or create new variables.

```
store_data %>%
  mutate(store_spend = store_spend / 100)
```

- Note how we can overwrite variables in a data frame as well as objects if we use the same name.

Joining Data Frames

- In the simplest case, a common variable (like an ID) allows us to join two data frames.

```
sat_data <- read_csv("sat_data.csv")

crm_data <- store_data %>%
  left_join(sat_data, by = "id")
```

Data Summarization

- Data summarization (as part of an exploratory analysis) is all about discovery: What is your data saying?
  - Describe data with numerical summaries (i.e., statistics).
  - Visualize data with graphical summaries.
- How we summarize depends on the whether the data is discrete or continuous.
  - Discrete is also called qualitative or categorical.
  - Continuous is also called quantitative or numerical.
- We will use both dplyr and ggplot2 to summarize data.

Describing Discrete Data

- The simplest numeric summary for a discrete variable is a count.

  ```
  crm_data %>%
    count(gender)
  ```

- Now get a count by both gender and country to produce a "tidy" cross-tab.

Visualizing Discrete Data

- Perhaps the most popular package, ggplot2 uses a consistent grammar of graphics built with layers.
    1. Data – Data to visualize.
    2. Aesthetics – Mapping graphical elements to data.
    3. Geometries – Or "geom," the graphic representing the data.
- Let's plot our previous summary (note how `+` is different from `%>%`).

```
ggplot(crm_data, aes(x = gender)) +
  geom_bar()
```

- Let's visualize a second variable. Add the aesthetic `fill = drv`.
- The geom `position` argument is set to `"stack"` by default. Try `"fill"` instead.

Describing Continuous Data

- The simplest numeric summary for a continuous variable is a mean.

```
crm_data %>%
  summarize(avg_store_spend = mean(store_spend))
```

- Note that `summarize()` is a more general version of `count()`.
- Compute the mean of both `store_spend` and `sat_overall`.
- We can also compute the mode, median, variance, standard deviation, minimum, maximum, etc.

Visualizing Continuous Data

- Let's plot the distribution of `store_spend`.

  ```
  ggplot(crm_data, aes(x = store_spend)) +
    geom_histogram()
  ```

- Change the geom `bins` argument to `10`.
- Visualize the relationship between `store_spend` and `sat_overall`.

  ```
  ggplot(mpg, aes(x = hwy, y = cty)) +
    geom_point()
  ```

- As part of the aesthetic, `log(store_spend + 1)`.
- Play with the `size` and `alpha` geom arguments.
- Add a `geom_smooth()` layer.

Describing Discrete and Continuous Data

- Grouped summaries provide a powerful solution for describing a combination of discrete and continuous data.

```
crm_data %>%
  group_by(gender, country) %>%
  summarize(
    n = n(),
    avg_store_spend = mean(store_spend),
    avg_sat_overall = mean(sat_overall)
  ) %>%
  arrange(desc(avg_store_spend))
```

- Grouping by a discrete variable is equivalent to filtering the data separately for each value of the group variable.
- `count()` is a wrapper around a grouped summary with `n()`.

Visualizing Discrete and Continuous Data

- There are geoms specific to visualizing the relationship between discrete and continuous data, including `geom_boxplot()` and `geom_density()`.
- But there are a variety of other options to quickly create impressive visuals.

```
crm_data %>%
  ggplot(aes(
    x = log(store_spend + 1),
    y = sat_overall,
    color = gender
  )) +
  geom_jitter(size = 2, alpha = 0.5) +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ country) +
  ggtitle("Store Spend by Overall Satisfaction")
```

Tidy Data

- Tidy data is defined as follows:
    1. Each observation has its own row.
    2. Each variable has its own column.
    3. Each value has its own cell.
- Fourth (sort of), each table has one type of observation unit.
- This may seem obvious or simple, but this common philosophy is at the heart of the *tidy*verse.
- It also means we will often prefer long datasets to wide datasets.

Gather and Spread Columns

- The most common problem with messy data is when columns are really values.

```
online_data <- read_csv("online_data.csv")
```

- Use tidyr to `gather()` the columns into key-value variable pairs.

```
online_data <- online_data %>%
  gather(
    key = week,
    value = visits,
    week1_visit:week4_visit
  )
```

- Note how this makes a wide dataset long (or at least long*er*).
- If the data has the opposite problem (i.e., values that should really be columns), `spread()` key-value pairs into columns.

```
online_data %>%
  spread(key = week, value = visits)
```

Separate and Unite Columns

- If two (or more) values are in one column, `separate()` the values into two (or more) columns.

  ```
  online_data <- online_data %>%
    separate(year_mo, c("year", "month"))
  ```

- When two (or more) values should be in one column, `unite()` the values into one column.

  ```
  online_data %>%
    unite(year_mo, year, month)
  ```

Exercise

1. In consecutive lines of code, do the following.
   a. Join `crm_data` and `online_data` `by = "id"`.
   b. Filter the data to keep only observations in the `"US"` in `2014`.
   c. Select the `store_spend`, `online_spend`, and `gender` variables.
   d. Create `log_store_spend` and `log_online_spend` (remember to add 1).
   e. Assign this data frame to `crm_data` (i.e., overwrite the existing `crm_data`).
2. Create a plot of `log_online_spend` and `log_store_spend`.
   a. Assign `gender` to the `color` aesthetic.
   b. Add `geom_smooth(method = "lm", se = FALSE)`.
3. Model the relationships you've just visualized with:

```
lm(
  log_store_spend ~ log_online_spend + gender,
  data = crm_data
) %>%
  summary()
```

Summary

- Covered the basics of coding in R.
- Practiced some essential data manipulation functions from dplyr.
- Explored the flexibility of grouped summaries for describing data.
- Practice the basics of plotting with ggplot2.
- Discussed the philosophy of tidy data.
- Used four functions for cleaning data from tidyr.
- Conducted a complete analysis: wrangling, summarization, and modeling.

Resources

- *R for Data Science*
- DataCamp
- *R for Marketing Research and Analytics*