Data Manipulation and Visualization in R

Oliwier Przewlocki
Technologisches Gewerbemuseum
oprzewlocki@student.tgm.ac.at

Contents

1. Introduction

We continue from the “Introduction to the R-Programming Language” and look at ways of optimizing the syntax to perform already familiar data operations and discover ways of visualizing the data.

2. Data Manipulation

This chapter covers ways to simplify already discussedways of manipulating data using tools like **Dplyr** (manipulating) and **Tidyr** (cleaning)

2.1. Dplyr

First, an installation using `install.packages('dplyr')` and an activation using `library(dplyr)` is required.

To allow for an easy way of showcasing the power of dplyr, the `nycflights13` package needs to be installed aswell using the same commands.

2.1.1. filter() & slice()

Allows for subset row selection. It’s easier than the built-in subset function we’ve been using. To filter, we just pass in the data as the first argument and then any number of arguments that specify each column condition:

```
filter(flights, month==11, day==3, carrier=='AA')
```

The slice function allows for positional selection:

```
slice(flights, 1:10) # outputs the first 10 rows
```

2.1.2. arrange()

Allows for ordering and has a very similar structure to the `filter()` function.

```
arrange(flights, year, month, desc(day)) # first order by year, then month and then by day but in descending order.
```

2.1.3. select() & rename()

Selects a defined set of columns:

```
select(flights, carrier, day) # selects the carrier and the day columns.
```

And `rename()` quickly renames the columns (newname = oldname)

```
rename(flights, newdayname = day)
```

2.1.4. distinct()

Selects distinct row values. Powerful to use in combinations such as with `select()`

```
distinct(select(flights, carrier)) # selects the distinct carriers.
```

2.1.5. mutate() & transmutate()

Creates new columns as functions of existing columns

```
mutate(flights, new_col = arr_delay-dep_delay)
```

If you only want the newly created columns, and not the entire data frame, use `transmutate()`

```
transmutate(flights, new_col = arr_delay-dep_delay)
```

2.1.6. summarise()

Summarizes a column using an aggregate function

```r
summarise(flights, avg_air_time=mean(air_time, na.rm = TRUE ))
```

2.1.7. sample_n() & sample_frac()

Sample_n() returns a random set of n-samples out of a data frame

```r
sample_n(flights, 10) # 10 random sample rows
```

Sample_frac() returns a percentage of data. The range goes from 0 (0%) to 1 (100%)

```r
sample_frac(flights, 0.3) # returns 30% of data.
```

2.2. The Pipe Operator

It allows better readability. An alternative to the pipe operator is nesting, which is barely readable beyond a certain point. You can also try to create multiple variables and assign them one by one, but then you're wasting memory space. The pipe operator allows you to chain an output of a function to the input of another function in a readable and clear way.

```r
# nesting
result <- arrange(sample_n(filter(df, mpg>20),size=5),desc(mpg))

# multiple assignments
a <- filter(df, mpg>20)
b <- sample_n(a, size=5)
result <- arrange(b, desc(mpg))

# pipe operator
result <- df %>% filter(mpg>20) %>% sample_n(size=5) %>% arrange(desc(mpg))
```

2.3. Tidyr

First perform the installation & setup using `install.packages('tidyr')` and `library(tidyr)`. There's also a complimentary package that needs to be installed called `data.table` using the same commands. This package is very similar to the built-in data frames, although it increases computation speed substantially.

2.3.1. gather() & spread()

A prerequisite to the `gather()` and `spread()` functions is the knowledge of the wide and long formats. Long formats have repeating first columns, and wide formats have values that don't repeat in the first column.

The gather function converts a table into a long format and the spread function converts a table into a wide format. That's it.

`gather()` collapses a slice of columns into a key-value pair (the key is the column name and the value is the row data value.):

```r
gather(df, key, value, Qtr1:Qtr4)
```

`spread()` widens the key-value pair passed into it to individual tables. The unique keys become columns and the values become the row values.

```r
spread(df, key, value)
```

2.3.2. separate()

Separates a single column into multiple columns. As an example, there is a column that has data like “a-x” and it’s required of us to separate “a” and “x” into their own respective columns

```r
separate(data=df, col= col.name, into c("abc", "xyz"), sep = '-')
```

2.3.3. unite()

Pastes multiple columns into one.

```r
unite(separated.df, new.joined.col, abc, xyz)

unite(separated.df, new.joined.col, abc, xyz, sep = "---")
```

3. Data Visualization

This chapter focuses on visualizing data using the **ggplot2** library.

3.1. Overview of ggplot2

The library is built on layers, namely the “**Data**” (the raw data), the “**Aesthetics**” (specify the columns and features you want to display) and “**Geometries**” (the type of plot) as the main layers.

Then there are other layers, such as “**Facets**” (multiple plots), “**Statistics**”, “**Coordinates**” and “**Themes**”

There exists a cheat sheet for `ggplot2` that explains the major aspects in around 2 pages. It can be found here.

3.2. Histograms

This section requires both the ‘ggplot2’ and ‘ggplot2movies’ libraries to be installed and activated.

```r
 # DATA & AESTHETICS
 pl <- ggplot(movies, aes(x=rating)) # rating is a column in the movies data frame

 # GEOMETRY
 pl2 <- pl + geom_histogram(binwidth = 0.2, color='red', fill='pink', alpha=0.8)
 print(pl2) # this actually displays the plot

 pl3 <- pl2 + xlab('Movie Rating') + ylab("Count") # x & y labels
 print(pl3 + ggtitle("MY TITLE"))


 # CUSTOM COLOR GRADIENT
 pl2 <- pl + geom_histogram(binwidth = 0.2, aes(fill=..count..)) # the higher the count the more blue it is.
```

3.3. Scatterplots

```r
 df <- mtcars

 # DATA & AESTHETICS
 ggplot(df, aes(x=wt,y=mpg))

 # GEOMETRY
 print(pl + geom_point(size=hp)) # size is based on the 'hp' column

 # FACTOR
```

```r
pl + geom_point(aes(size=factor(cyl))) # factor labels the 'cyl' column as categorical (not continuous) because it can only have either 4, 6 or 8 cylinders, not 5 or 7.

# SHAPE
pl + geom_point(aes(shape=factor(cyl))) # this assigns a different shape for different cylinder sizes, like a square or a triangle.

# CUSTOM COLOR GRADIENT
pl <- pl + geom_point(aes(color=hp), size=5)
pl + scale_color_gradient(low='blue', high='red')

# ATTENTION: You can only use column values to define shapes, colors, etc. inside of the aes() block.
```

3.4. Barplots

```r
df <- mpg

pl <- ggplot(df, aes(x=class))

print(pl + geom_bar(aes(fill=drv), position="dodge")) # The custom fill automatically creates a stacked bar plot. The position argument set to dodge makes the stacked bars appear next to eachother. There's also "fill" that makes it a normalized area graph.
```

3.5. Boxplots

```r
df <- mtcars

ggplot(df, aes(x=factor(cyl), y=mpg))
print(pl + geom_boxplot(aes(fill=factor(cyl))) + coord_flip()) # the coord_flip() flips the coordinate axes. It essentially rotates the plot -90°.
```

3.6. 2 Variable Plotting

```r
pl <- ggplot(movies, aes(x=year, y=rating))
print(pl + geom_bind2d() + scale_fill_gradient(high="red", low="green")) # creates a 2d-bin-chart. The bins change color based on their count. (Essentially a heatmap)
```

There's also a `hexbin`, which creates a heatmap where each bin is a hexagon. For that, install the `hexbin` package.

```r

# HEXBIN
pl <- ggplot(movies, aes(x=year, y=rating))
pl2 <- pl + geom_hex()
print(pl2)

# DENSITY PLOT
pl2 <- pl + geom_density2d()
print(pl2)
```

3.7. Coordinates Faceting

`ggplot2` allows for the changing of coordinate systems and the adjustment of their paramters like x-lims and y-lims:

```r
  pl <- ggplot(mpg, aes(x=displ, y=hwy)) + geom_point()

  # STANDARD COORDINATES
  pl2 <- pl + coord_cartesian(xlim = c(1,4), ylim = c(15,30))

  # FIXED RATIO COORDINATES
  pl2 <- pl + coord_fixed(ratio = 1/3) # Default is 1:1.

  # Consult the cheat sheet for more coordinate types.

  print(pl2)
```

Faceting allows for the placement of multiple plots next to eachother.

```r
  pl <- ggplot(mpg, aes(x=displ, y=hwy)) + geom_point()

  print(pl + facet_grid(. ~ cyl)) # we seperate the mpg plot (on the x-axis) into 3 subplots each seperated by the cylinder column. So there's one plot for cars with 4 cylinder, one for cars with 6 cylinder, etc.

  print(pl + facet_grid(drv ~ .)) # this separates the plot along the y-axis.

  # The general syntax is "what you want to faucet by the y-axis, tilde symbol, and then what you want to facet by on the x-axis." A dot means "everything else".

  # So the x and y separations can be mixed:
  print(pl + facet_grid(drv ~ cyl)) # this separates the plot along the y-axis.

```

3.8. Theming

You can set a theme by either setting it globaly using the `theme_set()` function or by just adding it when printing:

```r
 theme_set(theme_minimal()) # Setting the theme globally

 print(pl + theme_dark()) # Assigning the theme individually
```

To get even more themes, install the `ggthemes` library.

3.9. Data Visualization Assignment Solution

```r
    df <- fread("/home/ren/coding/r/R-Course-HTML-Notes/R-Course-HTML-Notes/R-for-Data-Science-and-Machine-Learning/Training Exercises/Capstone and Data Viz Projects/Data Visualization Project/Economist_Assignment_Data.csv", drop=1)
head(df)

  pl <- ggplot(df, aes(x=CPI, y=HDI)) + geom_point(shape=1, size = 3, aes(color=Region))

  pl2 <- pl + geom_smooth(aes(group=1), method = lm, formula = y ~ log(x), se = FALSE, color="red")

  pointsToLabel <- c("Russia", "Venezuela", "Iraq", "Myanmar", "Sudan",
                     "Afghanistan", "Congo", "Greece", "Argentina", "Brazil",
                     "India", "Italy", "China", "South Africa", "Spane",
                     "Botswana", "Cape Verde", "Bhutan", "Rwanda", "France",
                     "United States", "Germany", "Britain", "Barbados", "Norway", "Japan",
                     "New Zealand", "Singapore")

  pl3 <- pl2 + geom_text(aes(label = Country), color = "gray20",
                         data = subset(df, Country %in% pointsToLabel),check_overlap = TRUE)

  print(pl3 + theme_bw() + scale_x_continuous(name = "Corruption Perception Index (CPI)", limits = c(1, 10), breaks = 1:10 ) + scale_y_continuous(name = "Human Development Index, 2011 (1=Best)", limits = c(0, 1), breaks = seq(0, 1, 0.1)) + ggtitle("Corruption and Human development"))
```

3.10. Interactive Visualization with Plotly

Plotly allows for the interactive visualization of data in multiple languages like R, Python or Matlab. It's entirely open source, free and self-hosted.

A recent addition to their package is the ability to convert `ggplot2` plots into interactive plotly plots directly inside of the R environment.

To get started, install `plotly` activate it.

```r
 pl <- ggplot(mtcars, aes(mpg, wt)) + geom_point()

 gpl <- ggplotly(pl) # now you can zoom in and get more informationwithin the plot.
```

Their ggplot2 plotly documentation has alot of great tutorials on how to create various types of plots that can be accessed here.

3.11. Moneyball Assignment Solution

```r

  batting <- read.csv('/home/ren/coding/r/R-Course-HTML-Notes/R-Course-HTML-Notes/R-for-Data-Science-and-Machine-Learning/Training Exercises/Capstone and Data Viz Projects/Capstone Project/Batting.csv')

  head(batting)

  str(batting)

  batting$BA <- batting$H / batting$AB

  tail(batting$BA, 5)

  batting$OBP <- (batting$H + batting$BB + batting$HBP) / (batting$AB + batting$BB + batting$HBP + batting$SF)

  batting$SLG <- ((batting$H - batting$X2B - batting$X3B - batting$HR) + (2 * batting$X2B) + (3 * batting$X3B) + (4 * batting$HR)) / batting$AB

  str(batting)

  sal <- read.csv("/home/ren/coding/r/R-Course-HTML-Notes/R-Course-HTML-Notes/R-for-Data-Science-and-Machine-Learning/Training Exercises/Capstone and Data Viz Projects/Capstone Project/Salaries.csv")

  summary(batting)
  batting <- subset(batting, yearID >= 1985)
  summary(batting)

  merged <- merge(batting, sal, by = c("playerID", "yearID"))

  summary(merged)

# option 1
  lost_players <- filter(merged, playerID == "giambja01" | playerID == "damonjo01" | playerID == "saenzol01")

# option 2
  lost_players <- subset(merged, playerID %in% c("giambja01", "damonjo01", "saenzol01"))

  distinct(lost_players, playerID)
  count(lost_players, playerID)

  lost_players <- subset(lost_players, yearID == 2001)
  lost_players <- select(lost_players, H, X2B, X3B, HR, OBP, SLG, BA, AB)

  print(lost_players)

  sum(lost_players$AB) # 1469
  mean(lost_players$OBP) # 0.363

  replacement_players <- filter(merged, yearID == 2001, salary <= 5000000)
  replacement_players

  pl <- ggplot(replacement_players, aes(x = AB, y = OBP, name = playerID, salary = salary))
  pl <- pl + geom_point()

  ply <- ggplotly(pl, tooltip = c("x", "y", "name", "salary"))
  ply

  # I'd choose "berkmla01", "heltoto01" and "gonzalu01" as replacements for the lost players
```