Programming in Data science
Intro to dplyr Package

Asad Waqar Malik

Dept. of Information Systems
Faculty of Computer Science and Information Technology
University of Malaya
Malaysia

Oct 10, 2019

Lecture Outline

Introduction

- Sometime we may want to arrange the values in a certain way, drop or add some variables, or select only a subset of interesting cases.
- One of these tasks in data analysis is to select a subset of cases in the data that satisfy a given logical condition.
- That is exactly what the `select(..)`, `filter(..)`, and `pipeline` function does. It selects or filters the rows of the data table that meet certain criteria creating a new data subset.
- These functions are available in the dplyr package so, in order to use it, you need to have installed and loaded the dplyr package.

```r
install.packages("dplyr")  # install package
library(dplyr)             # include package
```

About dataset

⇒ The library called dplyr contains valuable functions to navigate inside the dataset.
⇒ In this lecture, we will use the Travel times dataset. The dataset collects information on the trip leads by a driver between his home and his workplace.
⇒ There are fourteen variables in the dataset, including:

- DayOfWeek: Identify the day of the week the driver uses his car
- Distance: The total distance of the journey
- MaxSpeed: The maximum speed of the journey
- TotalTime: The length in minutes of the journey

⇒ The dataset has around 200 observations in the dataset, and the rides occurred between Monday to Friday.

dplyr Package

- One handy feature with dplyr is the glimpse(..) function. This is an improvement over str(..).
- Use glimpse(..) to see the structure of the dataset and decide what manipulation is required.

```r
library(dplyr)
PATH <- "https://raw.githubusercontent.com/guru99-edu/R-↩
     Programming/master/travel_times.csv"
df <- read.csv(PATH)
glimpse(df)
```

dplyr Package

- Variable Comments needs further diagnostic – the observations of the Comments variable are only missing values.
- Count the observations equals to "" in the column comments from df.

```
> sum(df$Comments =="") # Sum the observations equalts to ↩
    "" in the column comments from df
[1] 181    # Output
```

Using Select(..)

- For our analysis, we don't necessarily need all the variables, and a good practice is to select only the variables you find relevant.
- We have 181 missing observations, almost 90 percent of the dataset. If you decide to exclude them, you won't be able to carry on the analysis.
- The other possibility is to drop the variable Comment with the `Select(..)`.

Select(..) format

```
> select(df, A, B ,C): Select the variables A, B and C ↩
    from df dataset.
> select(df, A:C): Select all variables from A to C from ↩
    df dataset.
> select(df, -C): Exclude C from the dataset from df ↩
    dataset.
```

Using Select(..)

⇒ select variables in different ways with `select(..)`.

- Lets use the third way to exclude the Comments variable.

```
> step_1_df <- select(df, -Comments)
> dim(df)
[1] 205  14  # Output

> dim(step_1_df)
[1] 205  13  # Output
```

⇒ The original dataset has 14 features while the step_1_df has 13.

Using filter(..)

- The `filter(..)` helps to keep the observations following a criteria.
- The `filter(..)` works exactly like `select()`, you pass the data frame first and then a condition separated by a comma:

filter(..) format

```
filter(df, condition)
arguments:
- df: dataset used to filter the data
- condition:  Condition used to filter the data
```

Using filter(..)

- Count the number of observations within each level of a factor variable.

table format

```
> table(step_1_df$GoingTo)
#table(): Count the number of observations by level. Note,↩
     only factor level variable are accepted.
#table(step_1_df$GoingTo): Count the number of of trips ↩
    toward the final destination.
#Output:
GSK  Home
105   100
```

Using filter(..)

filter observations

```
# Select observations
# Select observations
select_home <- filter(df, GoingTo == "Home")
dim(select_home)
Output:
[1] 100  14

# Select observations
# Select observations
select_work <- filter(df, GoingTo == "GSK")
dim(select_work)
Output:
[1] 105  14
```

Multiple criterion's `filter(..)`

- We can filter a dataset with more than one criteria.
- For instance, you can extract the observations where the destination is Home and occured on a Wednesday.

```
> select_home_wed <- filter(df, GoingTo == "Home" & ↩
    DayOfWeek == "Wednesday")
> dim(select_home_wed)
Output:
[1] 23 14
```

Pipeline

- The dataset requires a lot of operations, such as: importing, merging, selecting, filtering and so on.
- The dplyr library comes with a practical operator, %>%, called the pipeline.
- The pipeline feature makes the manipulation clean, fast and less prompt to error.
- This operator performs steps without saving intermediate steps to the hard drive.You can select the variables of interest and filter them, we have three steps:

Basic steps

```r
step_1 <- read.csv(PATH) # Step 1
step_2 <- select(step_1, GoingTo, DayOfWeek) # Step 2
step_3 <- filter(step_2, GoingTo == "Home", DayOfWeek == "↩
    Wednesday") # Step 3
```

Pipeline

- Let's use the pipeline operator %>% instead. We only need to define the data frame used at the beginning and all the process will flow from it.

Pipeline format – Basic syntax

```
New_df <- df %>%
step 1 %>%
step 2 %>%
...
arguments
- New_df: Name of the new data frame
- df: Data frame used to compute the step
- step: Instruction for each step
- Note: The last instruction does not need the pipe ↩
    operator %, you dont have instructions to pipe anymore
```

Pipeline

- You can create your first pipe following the steps enumerated above.

Pipeline steps

```
#Create the data frame filter_home_wed.It will be the ↩
    object return at the end of the pipeline
filter_home_wed <-

#Step 1
read.csv(PATH) %>%

#Step 2
select(GoingTo, DayOfWeek) %>%

#Step 3
filter(GoingTo == "Home",DayOfWeek == "Wednesday")
#identical(step_3, filter_home_wed)
```

arrange(..)

- The library dplyr has its sorting function. It works like a charm with the pipeline. The arrange(..) can reorder one or many rows, either ascending (default) or descending.

arrange(..) format

```
- arrange(A): Ascending sort of variable A
- arrange(A, B): Ascending sort of variable A and B
- arrange(desc(A), B): Descending sort of variable A and ↩
    ascending sort of B
```

```
# Sort by destination and distance
step_2_df <-step_1_df %>%
  arrange(GoingTo, Distance)
head(step_2_df)
```

dplyr Functions

- mutate() adds new variables that are functions of existing variables
- select() picks variables based on their names.
- filter() picks cases based on their values.
- summarise() reduces multiple values down to a single summary.
- arrange() changes the ordering of the rows.
- distinct, group_by_all, select_all, near and etc.

⇒ See dplyr documentation for all available functions.

Summary

Function	Format	Objective	Explanation
glimpse	check the structure of a df	glimpse(df)	Identical to str()
select()	Select/exclude the variables	select(df, A, B ,C)	Select the variables A, B and C
		select(df, A:C)	Select all variables from A to C
		select(df, -C)	Exclude C
filter()	Filter the df based a one or many conditions	filter(df, condition1)	Condition
arrange()	Sort the dataset with one or many variables	arrange(A)	Ascending sort of variable A
		arrange(A, B)	Ascending sort of variable A and B
		arrange(desc(A), B)	Descending sort of variable A and ascending sort of B
	Create a pipeline		

Filters

- This is how we do it:

```
model <- rownames(mtcars)        # Assign rownames
mtcars_n <- cbind(model,mtcars) # create new dataset with ↩
    mtcars and rownames
rownames(mtcars_n) <- c()        # remove row names from ↩
    mtcars_n
```

```
head(mtcars_n,4) #Just to make sure, lets take a look at ↩
    mtcars_n:
```

- The information is the same. We just changed the way car models are stored to make it easier to manipulate.

Filters

⇒ You are planning to buy a new car so you will practice the process of selecting car models with the characteristics you want using mtcars. Since you have a long commute, one of the most important characteristics for you is fuel efficiency.

- You want a buy a car with an average miles per gallon (mpg) greater than 25. The average mpg for each model is stored in the variable mpg. Thus, you will be selecting car models in mtcar_n that satisfy the criteria: mpg $>$ 25.
- These two elements are the necessary arguments for the filter function:

```
> filter(mtcars_n,mpg>25)
```

Explore yourself!

- The use of read_csv2(..) ????
- The use of read_delim(..) ????

```
#replace the ??? with NA
# delim is ""
d <- read_delim("z.txt", delim="", na="???")
```

Exercise

Questions

```
1  Select the first three columns of the iris dataset using ↩
       their column names. HINT: Use select().  select(iri, 1:3)
2  Select all the columns of the iris dataset except "Petal ↩
       Width". HINT: Use -.
3  Filter the rows of the iris dataset for Sepal.Length >= ↩
       4.6 and Petal.Width >= 0.5.  filter(iri,Sepal.Length >= 4.6, Petal.Width >= 0.5)
4  Create a new column called proportion, which is the ratio ↩
       of Sepal.Length to Sepal.Width. HINT: Use mutate().
```

new_iri <- mutate(iri, proportion = Sepal.Length / Sepal.Width)

Exercise

Factors

```
Load the gapminder data-set from the gapminder package. Save it to an object called gp. Check how many factors it contains.
library(gapminder)
gp <- gapminder
```

Exercise

Factors

```
Notice that one continent, Antarctica, is missing from the↩
    corresponding factor, add it as the last level of six↩
    .
>attributes(gp$continent)
???
```

Assignment

Question-1

```
Assignment:1
Your first assignment is to find a missing values in a ↩
    dataset, replace it with mean or median of that column↩
     or in case of non-numeric column, add "not-known" in ↩
    the missing cell.
Overall, your assignment should perform three steps:
a. Read the CSV dataset available for this assignment.
b. The dataset include "na", "NA" and "--" value, you need↩
     to clean that dataset, as mentioned above.
c. Store the new dataset into CSV format.

Your code should detect these values in any dataset.
Note: You would be interested to explore dplyr functions ↩
    such as mutate(...), transmute(...), replace(...).
Deadline: Oct 21, 2019.
```

Assignment

Question-2

```
The code to read the dataset is as follow:
survey <- read.table("http://www.andrew.cmu.edu/user/↩
    achoulde/94842/data/survey_data.csv", header=TRUE, sep↩
    =",")
Answer the following questions:
(a) How many survey respondents are from MISM or Other?
(b) What % of survey respondents are from PPM?
(c) Use $ notation to pull the OperatingSystem column from↩
     the survey data
(d) Do the same thing with [,] notation, referring to ↩
    OperatingSystem by name
(e) Repeat part (d), this time referring to ↩
    OperatingSystem by column number
Deadline: Oct 21, 2019.
```

Quiz - Instructions

Quiz-III

Quiz will be available at 8:00 PM and time duration is 10 Mins.
Your Google email should be working for participation.
You will get '0' marks, if you fail to submit your quiz within the time, due to any reason.